NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MAIL FILE ADMINISTRATION TOOL FOR A
MULTILEVEL HIGH ASSURANCE LAN

by

Richard Kip Rossetti

September 2000

| Thesis Advisor: | Cynthia E. Irvine |
| Second Advisor: | Paul Clark |

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE : A Mail File Administration Tool for a Multilevel High Assurance LAN	5. FUNDING NUMBERS
6. AUTHOR(S) Rossetti, Richard Kip	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)* Department of Defense official communications often require special protections to prevent accidental disclosure to unauthorized personnel. A Multilevel High Assurance LAN provides a framework for secure electronic communications, and obviates the need for multiple single level networks. A high assurance trusted computing base (TCB), allows untrusted commercial off-the-shelf (COTS) software, such as an Internet Message Access Protocol (IMAP) server, to run untrusted while access to the file system is mediated by the TCB. Control of creation and deletion of hierarchical structured objects, such as those in the file system, is based on the ability to write to the directory containing the object. For a mail server, this directory structure corresponds to a mailbox hierarchy. The mailbox hierarchy must be designed to allow users to read, create, and send mail at multiple levels. The purpose of this research is to develop a trusted process that automatically creates the mailbox hierarchy for any system user. A Mail File Administration Tool for a Multilevel High Assurance LAN allows administrators to easily set up IMAP-compatible mailboxes for each user. The tool assists in the management of the file structure and enables account administration for multiple LAN users and group accounts at multiple security levels.

14. SUBJECT TERMS Electronic mail, Multilevel, High Assurance, IMAP.			15. NUMBER OF PAGES 134
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std.
239-18

THIS PAGE INTENTIONALLY LEFT BLANK

A MAIL FILE ADMINISTRATION TOOL FOR A MULTILEVEL HIGH ASSURANCE LAN

Richard Kip Rossetti
Lieutenant, United States Navy
B.A., University of Colorado, Boulder, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
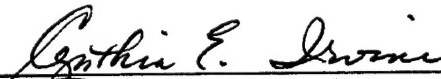
from the

NAVAL POSTGRADUATE SCHOOL
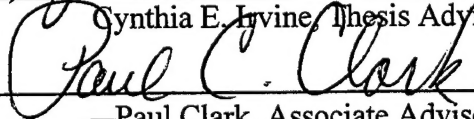### September 2000

Authors: _____

Richard Kip Rossetti

Approved by: _____

Cynthia E. Irvine, Thesis Advisor

_____

Paul Clark, Associate Advisor

_____

Daniel C. Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Department of Defense official communications often require special protections to prevent accidental disclosure to unauthorized personnel. A Multilevel High Assurance LAN provides a framework for secure electronic communications, and obviates the need for multiple single level networks. A high assurance trusted computing base (TCB), allows untrusted commercial off-the-shelf (COTS) software, such as an Internet Message Access Protocol (IMAP) server, to run untrusted while access to the file system is mediated by the TCB. Control of creation and deletion of hierarchical structured objects, such as those in the file system, is based on the ability to write to the directory containing the object. For a mail server, this directory structure corresponds to a mailbox hierarchy. The mailbox hierarchy must be designed to allow users to read, create, and send mail at multiple levels. The purpose of this research is to develop a trusted process that automatically creates the mailbox hierarchy for any system user. A Mail File Administration Tool for a Multilevel High Assurance LAN allows administrators to easily set up IMAP-compatible mailboxes for each user. The tool assists in the management of the file structure and enables account administration for multiple LAN users and group accounts at multiple security levels.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENT

The author would like to thank Prof. Irvine for the opportunity to work on this thesis, as well as her guidance, and support. I would also like to thank Dave Shifflet, and Paul Clark for their invaluable input and encouragement throughout the project.

Finally, I would like to thank my wife, my daughter, and our family and friends, whose support and patience made this possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

"The most often used technology on the Internet has been and is likely to remain, electronic mail [1]."

The Department of Defense (DoD) relies on electronic communications for a variety of operational and quality of life services. Electronic mail (e-mail) is being relied upon to aid workers in performing their jobs more reliably and efficiently. The key to its usefulness is its asynchronous nature. The individual to whom the message is being sent does not have to be presently available to receive the message. The sender can transmit the message, go about his business, and then, some time in the future, receive a reply.

The Internet-proposed standard Internet Message Access Protocol (IMAP) was developed as the successor to the current standard Post Office Protocol (POP)[1]. IMAP allows a user to send, receive, and edit messages from multiple locations and on multiple servers. IMAP was designed to permit manipulation of online messages as if they were actually stored on the user's machine. This storage scheme enables users to travel and have consistent access to the same mail repository. You can send receive and edit mail messages from anywhere you have a network connection. IMAP can access multiple mailboxes on the same or on different servers. An IMAP client may allow a user to see multiple mailboxes at the same time and move messages between them. Overall, the functionality of IMAP surpasses what is available in POP.

1

Without question, this is the future of Internet messaging. However, a security problem lies in the design of the server's file system. For example, the IMAP server may be collecting messages from multiple users. If a co-worker opens a message that contains malicious code, it has the potential to disrupt your service. Mandatory and Discretionary protection mechanisms need to be implemented on the IMAP server to protect its users from one another.

In a military environment, much of the information exchanged via electronic mail is classified. In this situation, the security of the server becomes even more important. DoD specifies four operating modes for computers handling classified information [2]. They are:

- ☐ Dedicated Security Mode: A system that handles information at a single classification level, and whose users are all cleared for that level of data.

- ☐ System High Security Mode: A system where only need-to-know protection is enforced between users. In this mode, all components connected to the system are classified at the highest level that information is being processed at, and all users are cleared up to the highest level.

- ☐ Multilevel Security Mode: The mode of operation in which two or more classification levels are being processed simultaneously and users do not share the same clearance levels for the information present on the system.

- ☐ Compartmented Security Mode: The security mode that allows two or more types of information compartments, or an information compartment and any other type of information on one system. In this mode, system access is secured to at least the Top Secret level. Users do not necessarily need to be authorized for all types of compartmented information. [3]

The Department of Defense Trusted Computer Systems Evaluation Criteria (TCSEC), DoD 5200.28-STD, states that one of its purposes is to:

> "provide DoD components with a metric with which to evaluate the degree of trust that can be placed in computer systems for the secure processing of classified and other sensitive information" [3].

Under these criteria, class B1 systems require labeled security protection. The typical military system running on UNIX or Windows NT does not apply labels. For example, Windows NT is a class C2 system that does provide Discretionary Access Control (DAC) and supports event auditing, but does not support Mandatory Access Control (MAC) and labels. Systems that are required to enforce MAC policy, and labeling for users with assorted security clearances often necessitate proprietary solutions that are difficult to maintain and expensive to upgrade. Another solution is required.

The partial goal of the Naval Postgraduate School Multilevel Secure Local Area Network (NPS MLS LAN) project is to develop a LAN that uses commercial-off-the-shelf (COTS) based software and systems with usable interfaces [4] [5] [6]. The LAN is based on a trusted server TCB to provide a secure multilevel mail-processing environment in which untrusted user processes can be securely executed while protecting the existing data. Protection includes prevention of unauthorized release, prevention of unauthorized modification, and protection from denial of service [7].

The first server implemented in the MLS LAN project was the mail server [8]. The trusted mail server consists of untrusted mail server instances mediated through a

3

high assurance TCB. The high assurance server was defined in earlier work [5] to be a commercial product already available on the Evaluated Products List with a Class B3 or higher rating based on the TCSEC [3]. The product selected was the Wang Government Services XTS-300 (XTS-300).

In previous research, the IMAP server has already been successfully ported to the XTS-300 [8]. The XTS-300 has a TCB that mediates all access to elements in its file system. The TCB enforces a hybrid MAC and DAC policy based on the Bell and LaPadula and Biba models [9] [10]. The mail spool and mail files stored on the XTS-300 are protected by the MAC and DAC policies.

The IMAP server was designed to access mail for single level systems. The MLS mail server will have mail files for some users at multiple levels. In order for a user to access his mail at multiple levels, something has to be done at the level of the file system. The first thing that is needed is a file structure that interfaces with the IMAP standards. The file structure contains the user mailboxes at the various secrecy levels. When a new user account is created, new mailboxes are needed. The mailboxes should follow the security policies of the multilevel file structure of the XTS-300 while enhancing usability for the customers by enabling them to be manipulated with regular COTS mail software.

In research by Eads [8], the only accessible mail stored on the XTS-300 was that at the user's current access class. The reason for this is that the IMAP server has a "home" mail directory from which it searches for readable mailboxes. The IMAP server implemented on the XTS-300 is instantiated per session level at client request. The code

is designed to find Inbox and other mail folders in a directory with a hard-coded path. Eads was able to change the "home" directory based on user session level. For example, if a user logs in at the secret level, the IMAP home is at a directory named secret. In essence, the IMAP "home" was a deflection directory. The only deflection directories visible to the user are those at his current session level. The native mail program on the XTS-300 also utilizes deflection directories. In a multilevel system, the MAC and DAC policies allow a user to read that are dominated by the current session level. This performance is desired at the client.

B. RESEARCH QUESTIONS

The objective of this research was to study the XTS-300 multilevel file system and determine potential file configurations that allow a user to access read down capability from an IMAP mail server running on the XTS-300. The candidate file system configuration were implemented and tested. This research was developed into an administration tool that allows mail administrators to duplicate the required mailbox hierarchy for system users. In the development of this tool, several questions were answered. Specifically:

☐ What is the best file structure for separating classified mail on a high assurance multilevel LAN?

☐ What file structures are supported by the XTS-300?

☐ How might pointers and/or links be used to support mail folder organization?

5

☐ What functionality needs to be included in the Mailbox Administration tool?

☐ Could a given file structure inhibit system performance and affect its ability to support usability?

☐ Does the administration tool and mail file structure support the administrative needs?

☐ Does the tool support a file structure that is not overly complex?

C.   ORGANIZATION OF STUDY

The Bell and LaPadula [9] and Biba [10] models are the basis for the security policies enforced by the Wang XTS-300 [11] [12] [13] [14] [15] [16]. The majority of material about the XTS-300 comes from the National Security Agency Final Evaluation report[11] [12] in which the XTS-300 was rated as a class B3 system under the standards of the TCSEC [3].

The background and implementation materials on the NPS MLS LAN comes from student thesis and published articles [4] [5] [6] [8].

IMAP was developed by Mark Crispin of the University of Washington. His draft RFC [17] provides most of the material on the IMAP protocol. Additional sources include a book by David Wood [1] on Internet mail and the RFC's for POP [18] and SMTP [19].

The paper starts with a background chapter on the two major components of the MLS LAN mail server, The XTS-300 system and the IMAP protocol server. At the end of this chapter is a background segment on the NPS MLS LAN. Chapter 3 discusses how

the XTS-300 interacts with the IMAP protocol and weighs in on what file hierarchy would be best suited for the server implementation. Then a description of a small experiment that was conducted on the file hierarchy followed by a description of the code required to make the chosen file hierarchy. Chapter 4 evaluates the tool based on criteria from previous MLS LAN research as well as criteria from the TCSEC. In the final chapter, some modifications to the tool are discussed as well as future work, conclusions and recommendations.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A.   WANG XTS-300

1.   System Overview

A high assurance base for a mail server provides a solution that allows COTS software to run safely in a high security environment. The high assurance Trusted Computing Base (TCB) is relied upon to enforce the underlying security policies while untrusted software runs isolated in a protection domain [5]. The COTS software, in turn, provides the ease of use, availability, affordability, selection of tools, and usable interfaces that cannot be equaled in proprietary software. This architecture allows users to take advantage of features and interfaces of familiar applications while working in a multilevel secure environment. The ultimate goal of this design is to provide secure mail processing to multiple workstations on a LAN that can fully utilize the protection mechanisms of a trusted server.

A High Assurance Multilevel Mail Server [8] is hosted on the Wang Government Services, Incorporated XTS-300 system [14]. The XTS-300, as evaluated by the National Security Agency (NSA), has a Class B3 security rating. Requirements for a Class B3 rating are specified in the TCSEC [3]. The XTS-300's TCB executes the COTS mail server in an untrusted domain such that the correctness of the operation of the mail server,

with respect to security, does not need to be established. In addition the COTS server, constrained by the TCB, does not compromise the XTS-300 Class B3 security rating [8].

A Class B3 rated system provides a TCB that enforces Mandatory Access Control (MAC), and Discretionary Access Control (DAC) policies. In addition, a Class B3 rated TCB provides a trusted path, an auditing and alarm mechanism, and separate administrator and operator roles. It also supports the principle of least privilege, meets software minimization requirements, provides layering, abstraction, and data hiding, and is resistant to penetration [3]. A system developer must base the TCB on a formal security model and provide a descriptive top level specification. The NSA team published the results of the evaluation in the Final Evaluation Report [12].

The XTS-300 system currently is hosted on an Intel Pentium based microcomputer processor. The Intel hardware has a protection level mechanism that is used in its self protection mechanism. The ring architecture consists of four privilege levels (PL0 –PL3), with PL0 being the most privileged. The STOP TCB uses privilege levels provided by the Intel architecture to enforce protection "Ring" mechanisms (Ring 0 – Ring 3). [12]

The Pentium has the ability to operate in four different modes; protected mode, system management mode, real-address mode, and virtual 8086 mode. The XTS-300 runs in real address mode at start-up and then switches to protected mode. The XTS-300 system does not utilize the Virtual 8086 or system management modes. The protected mode utilizes all of the protection features of the CPU, which includes its built in

10

instructions and protection ring architecture. The ring architecture is used to restrict access to segments, pages, and instructions. As seen in Figure 1, a non-TCB process makes use of a different ring architecture than a TCB process. A process is an active entity, which causes information to flow or changes a system state. A process may also be defined as a subject that accesses object data. Both trusted and untrusted processes utilize the same Ring 0 and Ring 1 assignment. [12]
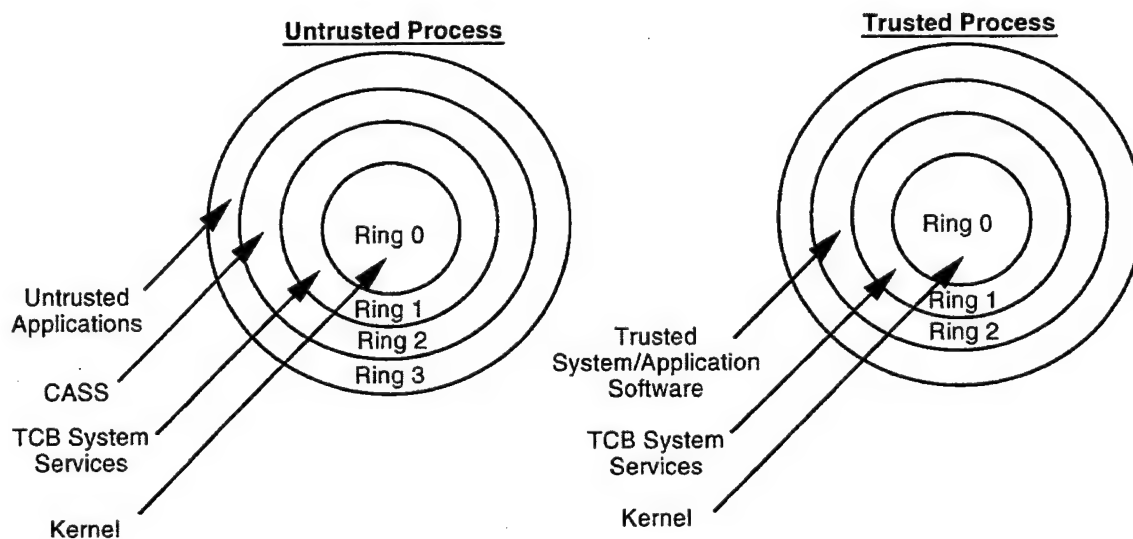
Figure 1. Privilege Level Ring Mechanism From Ref. [12].

The XTS-300 is broken into two major components: the TCB, and the Commodity Application System Services (CASS). The TCB includes the XTS-300 hardware and the trusted software portion of the STOP 4.2.2 operating system. Wang developed STOP 4.2.2 as a multilevel secure operating system. The operating system takes advantage of the ring architecture to help enforce its implementation of multilevel protection, and to provide self-protection for the TCB. To support all of the security

11

requirements the TCB and hardware mediate all requests to access data. STOP is broken into four components, the Security Kernel, the Trusted Computing Base System Services (TSS), the Trusted Software, and CASS. The four STOP components each occupy a separate hardware privilege level. [12]

As seen in Figure 1 the security kernel occupies Ring 0 for both trusted and untrusted processes. Ring 0 is the innermost and most privileged domain. A domain is an atomic program unit with its own address space. As a result, the security kernel is physically protected from other processes. The security kernel implements a variation of the reference monitor concept and enforces all Mandatory Access Control (MAC). The kernel also provides the basic operating system. When a process requests access to an object, the kernel performs access checks. If the access check passes, the kernel maps the object into the process's address space. The kernel provides input output (I/O) services and interprocess communication messaging mechanisms. The security kernel is part of every process's address space, however it cannot be called directly or modified by any process. The kernel can only be invoked via a kernel gate, hardware traps, faults, and interrupts. [12]

The TSS executes in Ring 1 and provides trusted system services for trusted and untrusted processes. It creates and loads trusted and untrusted programs into the operating system and supports user I/O. The TSS implements the hierarchical file system by converting and interpreting the flat kernel segment structure and it supports user I/O.

The TSS software also enforces Discretionary Access Control (DAC) policy to file system objects and segments. [12]

Trusted Software executes in the OSS domain at Ring 2. The user invokes trusted software by using a secure attention key (SAK). If the terminal is not currently logged in, the TCB requests a user to login. If the terminal is logged in, the TCB queries the user for a command to execute. When the trusted path is invoked, any currently executing software, either trusted or untrusted, no longer has access to the terminal. A process is considered trusted because it could violate security policy but is trusted not to. For example a trusted command called file system manager (**fsm**), allows an authorized operator or administrator to modify the access attributes of files. Some processes require privileges to perform their functions. A privilege is a controlled mechanism whereby a process may be authorized to bypass a system security policy in a controlled manner. An example of a trusted program privilege is DISCRETIONARY_ACCESS_EXEMPT, whereby a process is allowed to bypass DAC policies to modify an object. A complete list of privileges understood by STOP 4.4.2 is outlined in Appendix B. Figure 2 shows the organization of a trusted process running on the XTS-300. [12]
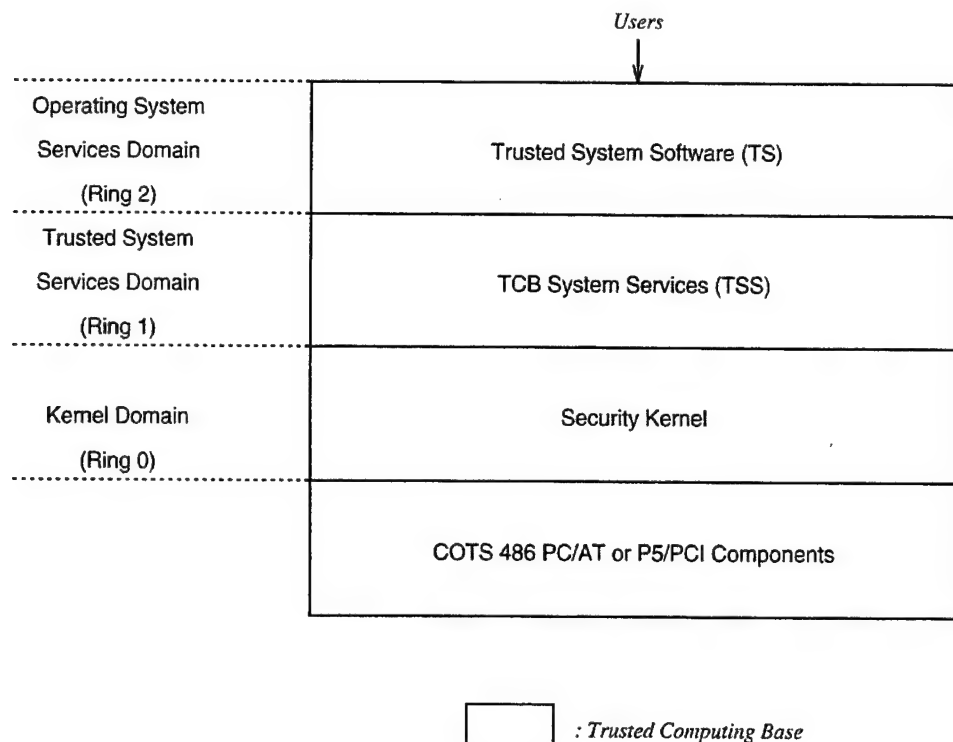
Figure 2. XTS-300 System Diagram- Trusted Process From Ref. [12].

CASS also executes in Ring 2. CASS is outside the TCB and does not involve trusted processes. CASS provides a UNIX-like interface for user-written applications. The purpose of CASS is to make the multilevel security execution environment transparent to software running in the application domain, which is Ring 3. Figure 3 shows the organization of an untrusted process running on the XTS-300. [12]
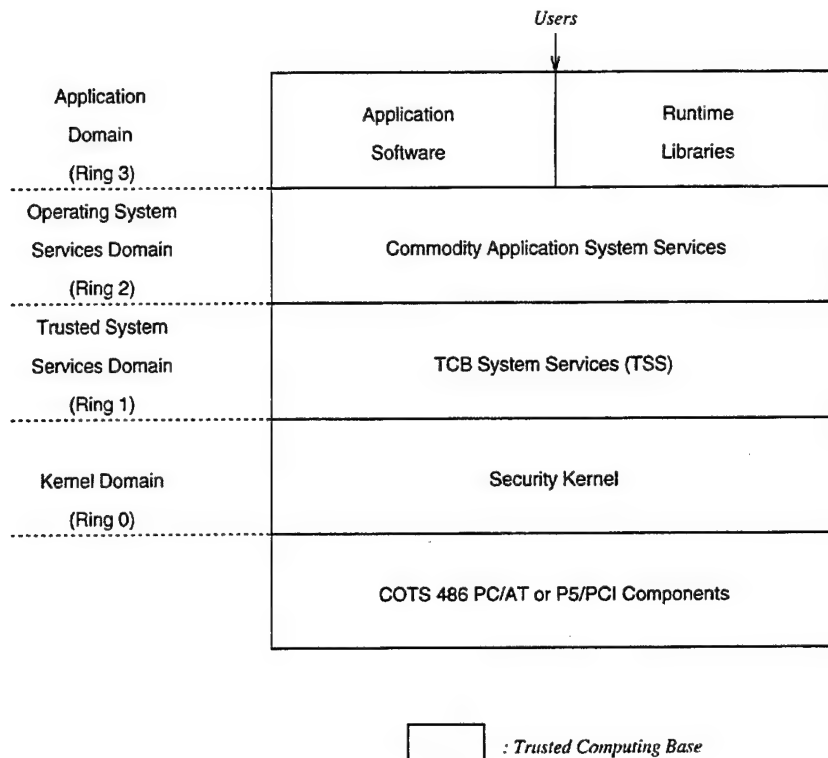
Figure 3. XTS-300 System Diagram- Untrusted Process From Ref. [12].

2. TCB Protection Mechanisms

As stated earlier, the TCB is comprised of the XTS-300 hardware and the trusted software portion of the STOP 4.2.2 operating system. The most important services provided by the TCB are its protection mechanisms [12]. These mechanisms enforce system policy, provide separation of address spaces, protect trusted code, and provide accountability for actions taken by subjects. The hardware protection mechanisms are established in the Intel hardware privilege architecture and help mediate processes accesses to objects.. The software mechanisms concentrate on the enforcement of policy and accountability. [12]

15

The STOP 4.4.2 operating system uses a hybrid security policy that combines the Bell and LaPadula model [9] for security and the Biba model [10] for integrity. The hybrid model states that conditions for both models must be met for a subject to be granted access to an object. STOP uses the mandatory and discretionary access control policies of the model to protect data on the file system.

The security kernel, through the use of MAC labels, enforces the Mandatory Access Control (MAC) Policy. Every subject and every object has a MAC label assigned to it. A MAC label is a combination of a sensitivity label and an integrity label. A sensitivity label includes one of 16 hierarchical sensitivity levels (sl) and from 0 to 64 non-hierarchical sensitivity categories (sc). An integrity label is comprised of one of 8 hierarchical integrity levels (il) and from 0 to 16 nonhierarchical integrity categories (ic). Figure 4 shows a MAC label representation with the internal storage format and information content.

MAC Label		
Sensitivity Level 8-bits		
16 hierarchical levels :		*0- 15*
Sensitivity Category 64 -bits		
64 nonhierarchical categories :		*0- 63*
Integrity Level 8-bits		
8 hierarchical levels :	*0-3*	*User Integrity*
	4	*OSS Integrity*
	5	*Operator Integrity*
	6	*(not predefined)*
	7	*Administrator Integrity*
Integrity Category 16-bits		
16 nonhierarchical categories :	*0- 7*	

Figure 4. STOP 4.4.2 MAC Label Representation After Refs. [11] and [16].

A dominance relationship is defined between labels in order to adjudicate subject access to objects. Given two MAC labels, the first is considered to "dominate" the second if the hierarchical level of the first is greater than or equal to that of the second, and if the category set of the first is a superset of the second. This comparison rule holds for both sensitivity and integrity labels. For sensitivity labels the subject must dominate the object, for integrity labels the object dominates the subject [12].

This dominance relationship supports the Bell and LaPadula model [9], and the Biba [10] model to make a MAC policy enforcement mechanism using four basic MAC rules. These rules are:

17

Simple Security Property: A subject may access an object for reading if the sensitivity label in the current MAC label of the subject dominates the sensitivity of the object.

Simple Integrity Property: A subject may access an object for reading if the integrity label of the object dominates the integrity label in the current MAC label of the subject.

Security *-Property: A subject may access an object for writing if the sensitivity label of the object dominates the sensitivity label in the current MAC label of the subject.

Integrity *-Property: A subject may access an object for writing if the integrity label in the current MAC label of the subject dominates the integrity label of the object. [12]

Since this is a combined policy, if a subject wants to read an object, both the simple security and integrity properties must be met. Likewise, to access an object for writing, both the security *- and integrity *- properties must be met. Control of creation and deletion of hierarchical structured objects, such as those in the file system, are based on the ability to write to the directory containing the object [12]. This important distinction will be discussed in the section on the XTS-300 file system.

Discretionary Access Control (DAC) policy specifies three allowed access modes between subjects and objects; read (r), write (w), and execute (x). In a manner similar to UNIX, STOP grants access based on a relationship between a named user and a named object. Each user has a unique identifier as well as a list of groups with which the user is associated. A subject, acting on behalf of the user, has the user and group identification(ID) entered into the Active Process Table Entry for the process. STOP supports the concept of real and effective user identification. Access checks are done

18

based on the current effective user and group identification. In addition, each object has

an Access Control List (ACL) that contains the following entries:

Owning user ID of the object, and the allowed access modes of that user. For a process, this user ID is interpreted as the effective user ID.

Owning group ID of the object, and the allowed access modes of that group. For a process, this group ID is interpreted as the effective group ID.

A list of up to seven other users or groups and their allowed access modes.

Allowed access modes for any other user or group that has not already been covered. [12]

For each ACL entry the access mode bits (r,w,x) are set. If an allowed access bit

is set for a particular ACL entry, the named user or group is allowed that permission upon

the object (subject to MAC policy). For example, an object named foo has the following

ACL representation:

Owning user:	Bob	rwx
Owning Group:	Student	r- -
Other user/Group	None	
Others		- - -

A process acting on behalf of Bob would have read, write and execute permissions on foo

(subject to MAC policy). A member of the Student group would have read permissions

on foo (subject to MAC policy). Every other user would not be able to access foo.

3.   File System Hierarchy

The TSS implements a hierarchical file system that is layered on top of the

security kernel's file system structure. The files are essentially labeled objects that are

19

containers for information. The file structure is hierarchical, meaning that files are stored in a "family tree" structure. The files may be one of four types: directories, files, device special files, or FIFO's [14]. At the top of the hierarchy, or "family", of each file system is a single directory known as the root. The root is the "parent" of the files created below it, the "children". An important characteristic of the XTS-300 file system is that the parent of an object is always readable at the access class of that object [14]. This implies that the root must have at the lowest sensitivity label and the highest integrity label of a given hierarchy.

The root must have the lowest sensitivity label because of the simple security property [9]. The root is the parent of all the subdirectories created below it in the hierarchy. The parent must be readable from the MAC level of the child subdirectory; therefore, any process operating at the sensitivity level of the child must dominate the sensitivity label of the parent. Likewise for the simple integrity property [10]. Any process operating at the integrity level of the child must dominate the integrity label of the parent.

Another important distinction of the file system is that control of creation and deletion of hierarchical structured objects is based on the ability to write to the directory containing the object [12]. Applying the hybrid security policy with all four MAC security policies, the implication is that a child must be created at the same MAC level as the parent. Applying the security *- property [9], a process, operating at the sensitivity level of a child, may write to the parent directory, if the parent's sensitivity label

dominates the child's. Since it was previously stated that the child's sensitivity label must dominate the parents, the only way to reconcile the discrepancy is for both parent and child to have the same sensitivity label. Likewise for the integrity *- property, a process operating at the integrity level of a child must dominate the integrity label of the parent. It was previously stated that the parent's integrity label must dominate the child's. In order to reconcile the discrepancy both the parent and child must have the same integrity label.

The hybrid security policy is so strict that, unless another mechanism is available to an administrator, only single level objects may be accessed. A single level file system does not take advantage of the extendibility and capacity of the hybrid security policy model. The hybrid security policy allows subjects to read objects that the subject's sensitivity label dominates. In this sense it enables "read down". The hybrid model also allows subjects to write to objects that the subject's integrity label dominates. This is the ability to "write up". Read down and write up are two characteristics of the file system the mailbox hierarchy was designed to take advantage of.

The XTS-300 gives administrators the ability to provide controlled exemptions to the access control rules normally enforced by the TCB to satisfy operational requirements. This is accomplished through the use of trusted subjects. Trusted subjects use the privileges outlined Appendix B to perform their necessary tasks. In the case of a file system, the trusted command **fsm** allows users with sufficient authorization, or administrators, to bypass the security *- property. With this privilege, a trusted process, acting on behalf of a subject may write to an object that has a lower sensitivity label.

This enables controlled "write down" capability. With this privilege administrators are able to create file hierarchies that are multilevel.

This is important in making the mailbox hierarchy. The mailbox hierarchy resides in Ring 1 in the TSS. The mailbox hierarchy is constructed of directories and files. Directories make up the parent nodes of the mailbox family tree. Directories may also be children of other directories. When a trusted subject is involved, the child directory may have a higher sensitivity label than the parent directory. The XTS-300 file system does not allow files to have children. Files are always child nodes, and, in the mailbox hierarchy, always have the same access class as their parent. For this reason, trusted processes are not needed to create, delete, or modify files in the mailbox hierarchy. Mailboxes are files. This means that untrusted applications running in the CASS environment are allowed to access mailboxes that have the same MAC label as the current session for reading, and writing. In a properly designed mailbox hierarchy, untrusted applications are also allowed to access mailboxes that are dominated by the current session for read only.

Another notable characteristic of the XTS-300 file system is that the parent node "knows" about, or is able to read the file name of all of its children. This applies regardless of the MAC label of the child node. The reason for this is that when a file or subdirectory is created, its name, segment ID, and file system ID are all stored in the directory segment of the parent. These three elements are always readable by the parent even though a process operating at the access class of the parent may not access the

22

object to read. What this means to the mailbox hierarchy is that root directories always know the names of their subdirectories. A process operating at a sensitivity level of unclassified may be able to see that the user has a mailbox named "top-secret". The unclassified process will not be able to read the directory named "top-secret", but some organizational need to know policies may be bypassed. The problem noted here is known as the "knowability" problem, and it cannot be solved with any capabilities available in the XTS-300 TCB.

B.   INTERNET MESSAGE ACCESS PROTOCOL (IMAP)

1.   Basic Internet E-mail

The "basic" mail system is a peered distributed client/server system built around the Simple Mail Transfer Protocol (SMTP) and its extensions. Clients send and receive mail by talking to servers, and the servers talk to one another. A client may send a message to one server that can deliver it to the recipient or transfer the message to another server. In addition to SMTP, four other elements help make up the basic mail system. They are the mail user agent (MUA), mail transfer agent (MTA), mail delivery agent (MDA), and the mail retrieval agent (MRA). [1]

The client in the client/server system is the MUA. MUA's are the programs used on a workstation to send and receive mail. MUA's used in the MLS LAN project include Microsoft Outlook, Netscape Communicator, and Pine. Many other products support IMAP and could have also been effectively used as MUA in this project.

23

The server is also called an MTA. An MTA is a collection of programs that transfer e-mail from one machine on the Internet to other MTA's. MTA's both send and store messages. MTA's utilize another set of programs to deliver messages to local mailboxes as well as remote MTA's. These programs are known as MDA's. Different MDA's perform different tasks. For example, a MTA would call on one MDA to perform delivery to a remote MTA, and call on another MDA to perform local delivery.[1]

If a user's mailbox is not on the same machine as his MUA, the MUA calls on another program to retrieve the messages. This program is the MRA. A MRA is really a server/client system as both MUA and MTA must cooperate. The MTA provides access to the user mailbox while the MRA retrieves the message to be displayed on the MUA. Post Office Protocol (POP) and IMAP are examples of MRA's. Figure 5 is a simplified example of the relationship that exists between these elements.
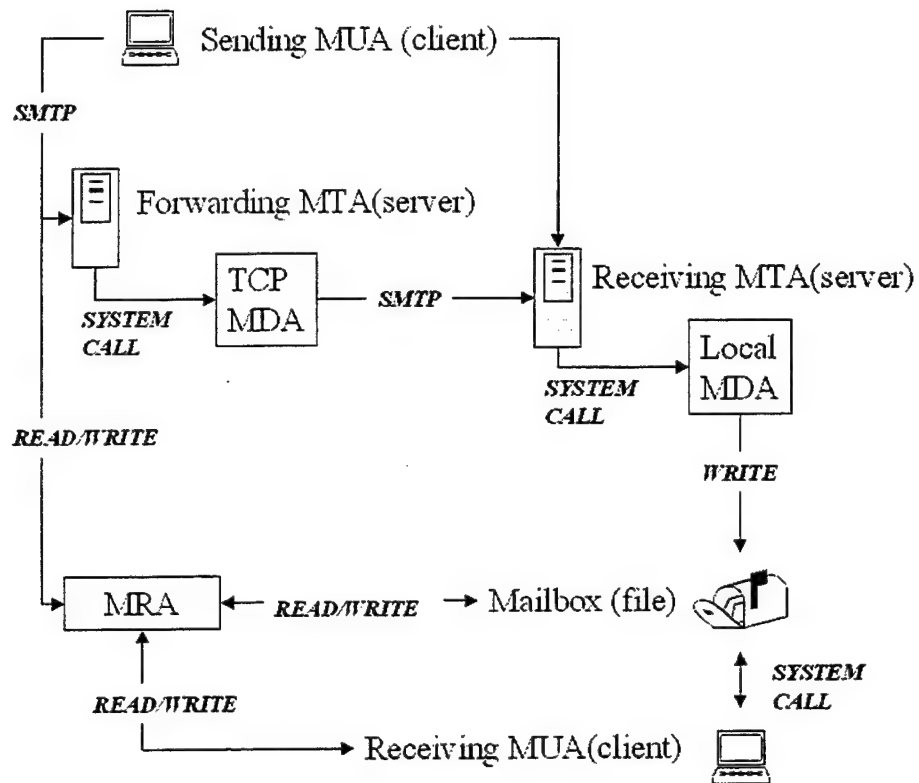
Figure 5. Simplified Internet Mail System After Ref. [1].

SMTP [19] is the standard for servers that move mail over the Internet. SMTP was designed when most users were directly accessing their mail from mail servers, either from a terminal connected to a mainframe, or a UNIX workstation that was designed to receive mail. If a MUA cannot reach a receiving MTA in a given amount of time, an error message is sent to the sending MUA. In current practice, most mail is accessed asynchronously. Users are connected to the Internet only part-time through a telephone line. Immediate delivery of messages is not guaranteed, so the design of the mail system required extension. Mail needed to be delivered to a MTA that is permanently connected

to the Internet and always reachable. Mail is stored on a server and can be queued for later delivery to a user's MUA, thus avoiding error messages if a user is offline. [1]

Now messages written to the server-side mailbox need to be read by a machine across the Internet. In addition, users want to access their mail from multiple locations. These problems are solved by use of MRA's. An MRA allows "disconnected" users to receive and access mail stored on their server from any location across the Internet. Multiple servers and multiple mailboxes can be accessed from any terminal connected to the Internet. Figure 6 shows how mailboxes are accessed by a MRA. [20]
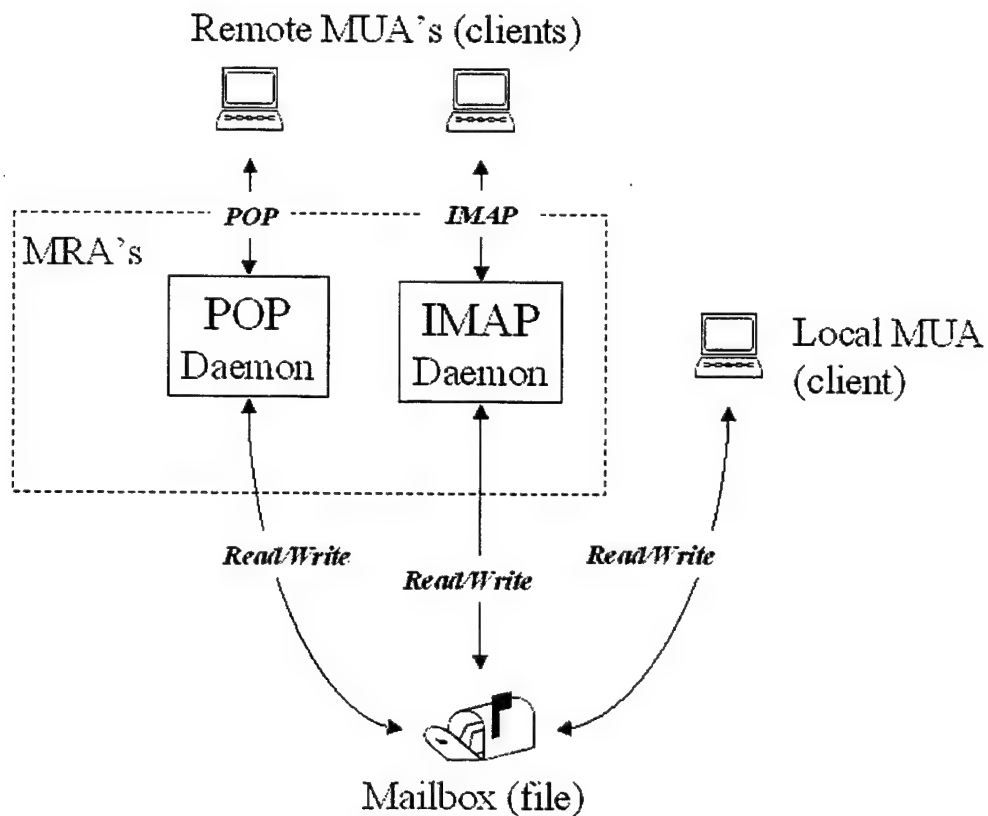
Figure 6. Accessing a Mailbox With a MRA After Ref. [1].

2.   IMAP -VS- POP

The Internet has two MRA protocols: the older and simpler POP and the newer and more complex IMAP [1]. POP supports "offline" mail processing where mail is stored on a server until a client requests it. It is described by RFC 1939 *Post Office Protocol- Version 3* [18] and is considered a standard mail protocol. IMAP was developed as the successor to POP and it supports both "offline" and "online" processing of messages. IMAP is an Internet-proposed standard and is defined in RFC 2026 *Internet Message Access Protocol – Version 4, Revision 1(IMAP4rev1)* [17].

POP was designed as a temporary storage point until the messages could be downloaded to a user's workstation or saved on the user's file space [1]. POP supports "offline" mail processing where mail is stored on a server until a client requests it. Once the client requests the mail the mail is forwarded and stored on the client machine. Mail stored on the server can be left on the server or deleted. All further mail processing takes place on the client machine. POP is a simple protocol in that it was designed as a download-only protocol. This means it has two options: download and delete, or just download. There is no provision for performing any other actions on the mail while it is stored on the server. It is also difficult to access messages from multiple locations. Mail tends to be scattered as it is stored on the various computers and deleted from the server. Even if a user does not delete the mail from the server, changes made to a message in one location will not be reflected when the user accesses it on another computer. In addition,

a user cannot access multiple mailboxes on a server without having multiple user names and passwords. [20]

IMAP was designed to access and manipulate remote messages as if they were local [1]. For this reason IMAP is more complex than POP and has a richer set of commands. This also means that client server interaction is more elaborate in IMAP than in POP. IMAP can do "offline" processing like POP. In offline mode, mail is stored on the server and all processing takes place on the client. IMAP can also perform "disconnected" processing whereby messages can be downloaded onto a client and edited. Later the user can synchronize the client and server, whereby the message stored on the computer replaces the message on the server. An edited message stored on a server can also replace a message stored on a client. The strength of IMAP however, is in its "online" processing. The client can then use the various IMAP commands to retrieve, store, save, delete, and create mail messages directly on the server. This allows a user to interactively access multiple mailboxes from multiple clients [20]. This model complements the design of the MLS LAN where thin client machines are connected to a server via a LAN. The thin clients activate MUA's that use IMAP4rev1 to access and update mail stores held on a high assurance multilevel server. The server uses a high assurance TCB to mediate access control and since all the mail messages are stored on the high assurance server, the mail stores are secure. IMAP is able to provide access to information stores at multiple levels without compromising information security [8].

3.   IMAP Commands

Similar to POP, IMAP4rev1 defines four separate states from which a client may issue commands. These states are: Nonauthenticated State, Authenticated State, Selected State, and Logout State.   Commands are organized by state in which a command is permitted. In some cases, commands are permitted in multiple states. [1] [17]

The following commands are allowed in any state:

- CAPABILITY: This command takes no arguments and returns a listing of capabilities that a server supports.

- NOOP: This command takes no arguments and does nothing. The NOOP command always succeeds. It is used to periodically poll for new messages or status updated during a period of inactivity.

- LOGOUT: Takes no arguments, and informs the server that the client wishes to close the connection. [1] [17]

Upon initial connection, the system starts in the Nonauthenticated State. IMAP will not allow most commands until the client is authenticated.   In addition to the commands available at any state, the following commands are   allowed in the Nonauthenticated State:

- AUTHENTICATE: This command takes as an argument an authentication mechanism name. This is a name of a standard authentication mechanism which is supplied to the client by the server in response to the CAPABILITY command. If the server supports this mechanism, which it should since it is listed in its

29

capability set, an authentication protocol follows. Upon completion of the authentication protocol, the client changes to the authenticated state. If the authentication mechanism fails, or is not supported by the server, there is no response. The client may try another authentication mechanism. This will continue until, as a last resort, the LOGIN command will be used to authenticate the client.

- LOGIN: This command takes as its augments the user name and password of the client user. The name and password are passed as plaintext so anyone monitoring the network traffic can obtain plaintext passwords. For this reason, the LOGIN command is used as a last resort. [1] [17]

Upon entering the Authenticated State, eleven commands are available to the client that allows creation, deletion, and examination of mailboxes. Two of these commands enter the client into the Selected State. In addition to the commands available in any state, the following commands are available in the authenticated state.

- SELECT: This command takes a mailbox name as an argument. If the mailbox is selectable, the messages in that mailbox can be accessed. Once a mailbox is selected, the session is placed in the Select State.

- EXAMINE: This command takes a mailbox name as an argument. It selects the mailbox as with the SELECT command but in a read-only mode. This command places the session in the Select State.

30

- CREATE: This command takes a mailbox name as an argument. It creates a new mailbox with a specified name.

- DELETE: This command takes a mailbox name as an argument and deletes the mailbox with the specified name.

- RENAME: This command takes an existing mailbox name and a new name as arguments and renames the mailbox.

- SUBSCRIBE: This command takes a mailbox name as an argument and puts the name into a list of "subscribed" mailboxes. This list is maintained in a file that can be accessed through the LSUB command.

- UNSUBSCRIBE: This command takes a mailbox name as an argument and deletes the name from the "subscribed" list.

- LIST: This command takes a reference name, and a mailbox name as its arguments. IMAP4rev1 supports wildcards as argument input. This command returns a subset of available mailboxes based on the input.

- LSUB: This command takes a reference name, and a mailbox name as its arguments with possible wildcards. This command returns a subset of the list of mailboxes on the "subscribed list".

- STATUS: This command takes a mailbox name and a status data name as an argument. The indicated status of a mailbox is returned which can include: the number of messages, number of recent messages, the next unique

identifier(UID) of the mailbox, the UID validity value of a mailbox, and the number of messages that do not have the \Seen flag set.

- APPEND: This command takes the mailbox name, flags, data time string, and message literal and appends the message to the end of the named mailbox. [1] [17]

The next state is the Selected State. It is entered with the SELECT or EXAMINE command from the Authenticated State. The Selected State gives the client access to commands that allow editing of specified messages. Commands from the authenticated state are available in addition to the following:

- CHECK: Takes no arguments and requests any server specific housekeeping be performed on the currently selected mailbox. If the server has no housekeeping, this command is equivalent to a NOOP.

- CLOSE: Takes no arguments. This command permanently deletes all messages that have the \Deleted flag set and returns to the Authenticated State.

- EXPUNGE: Takes no arguments. This command permanently deletes from the currently selected mailbox all messages that have the \Deleted flag set. This command returns a response for each expunged message and remains in the Selected State.

- SEARCH: Takes a (optional) [CHARSET] specification and one or more search criteria as arguments. It searches the current mailbox for messages that

32

meet the passed in criteria. 36 search keys are identified for IMAP4rev1 and
are listed in RFC 2026. [17]

- FETCH: Takes a message set and message data item names as arguments. It
  retrieves data items associated with a message in the mailbox.

- STORE – Takes a message set, message data item name, and the value for the
  massage data item as arguments. It changes the current message data item to
  the given value.

- COPY: Takes a message set and a mailbox name as an argument. It copies the
  specified messages to the end of the mailbox.

- UID – Takes the command name and command argument as arguments. It
  acts like the argument you pass in (FETCH, COPY, STORE , or SEARCH)
  except a Unique identifier is returned for each. [1] [17]

C.   SERVER IMPLEMENTATION

The specifications for a High Assurance Multilevel Mail Server are detailed by
Eads [8]. In essence, a server should implement the standard IMAP. Modifications to
IMAP should not affect the successful execution of COTS MUA's, and the server must
provide storage locations for data of various sensitivity levels.

An IMAP server is a form of MRA. The current version is IMAP4rev1. An
IMAP server has a TCP application daemon that listens for IMAP requests on TCP port
143. MUA's that wish to connect to a server try to connect to this port. The server
responds by spawning a new instance of the IMAP server on a new port number and

33

communicates this port number to the client. In the MLS LAN implementation the user is pre-authenticated by the XTS-300 so the IMAP server starts in the Authenticated State. The IMAP server instance is spawned at the same MAC level as the current session. The server then sends a banner greeting to the MUA and awaits further commands.

In the authenticated state, the untrusted server is running at the level of the current session. That is, the process that is operating on behalf of the user, is running at the MAC level that the user pre-authenticated with the XTS-300. The untrusted software, which includes the IMAP4rev1 commands, is allowed to access the file system below it through the TCB. The TCB enforces its protection mechanisms on the file system. Even though untrusted commands are being issued, the security kernel mediates all accesses. In this manner, IMAP commands are allowed to operate on the mailboxes which are held in the XTS-300 file system. Figure 7 shows an implementation of the NPS MLS LAN.
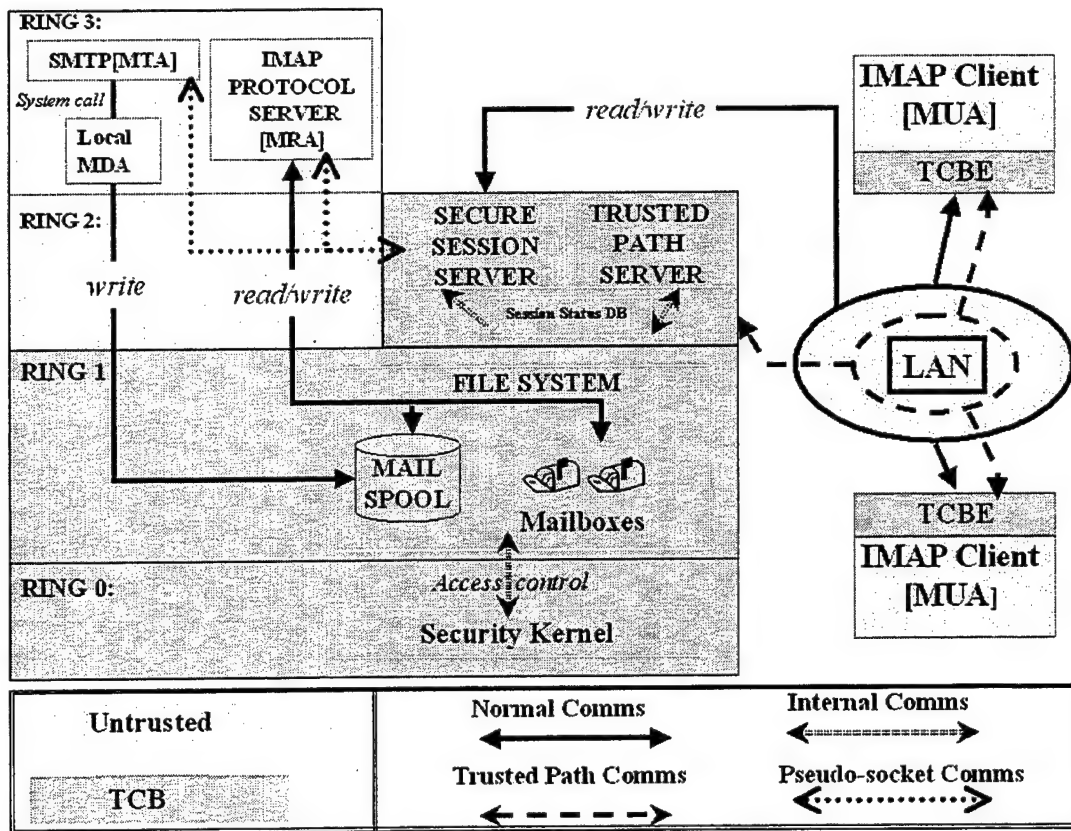
Figure 7. High Assurance Multilevel LAN for Commercial PC's After Refs. [6] and [16].

THIS PAGE INTENTIONALLY LEFT BLANK

III. DEVELOPING AN ADMINISTRATION TOOL

A. DISCUSSION

The Internet Message Access Protocol, Version 4rev1 (IMAP4rev1) allows a mail client such as *Pine*, *Outlook*, or *Netscape*, to access and manipulate remote message stores or "mailboxes". The client, or Mail User Agent (MUA), is loaded onto an individual PC to allow a user to send and receive mail. For the NPS MLS LAN, the IMAP Protocol server resides on the XTS-300 in the untrusted CASS environment (Ring 3). The IMAP protocol server, acting as a Mail Retrieval Agent (MRA) can access mailboxes on the file system (Ring 1) or the mail spool (Ring 1).

In this configuration, IMAP4rev1, acting as a subject, is able to retrieve objects on the XTS-300 that are stored at the access class of the current session. A remote MUA, using IMAP, is then able to read, create, modify, and delete these objects directly on the XTS-300 file system. The MAC policies of the XTS-300 file system also allow IMAP4rev1 to read objects whose MAC Labels are dominated by the MAC level of the current session. The objects on the XTS-300 can be either directories, which contain information about other objects, or files which contain data.

IMAP4rev1 interprets a path name as a mailbox. For example, 'bob/mbox' is the name of a mailbox. The 'bob' portion is a directory, while the 'mbox' is a file. The 'mbox' file has a special format that enables mail messages to be appended to it. The mbox format is also known as the "Unix format" and a variation of it is used on the NPS IMAP

37

server. On an IMAP server, only mailboxes contain messages. The mailbox name can be separated on the XTS-300 file system into directories and files. The directory 'bob' cannot store messages, only information about the file 'mbox'. The file 'mbox' contains the data fields that make up the e-mail messages for user Bob.

A directory path denotes a mailbox naming hierarchy through which a user's e-mail can be organized. A hierarchical relationship exists between mailboxes if the mailboxes share a common root. The mailbox names are read left to right with a single character denoting separate levels of a hierarchy. For example, 'bob/newsgroup/mbox' and 'bob/work/mbox' are the names of two mailboxes that are part of the same mailbox hierarchy because they share a common root. Figure 8. shows how this mailbox hierarchy would be represented at a MUA.
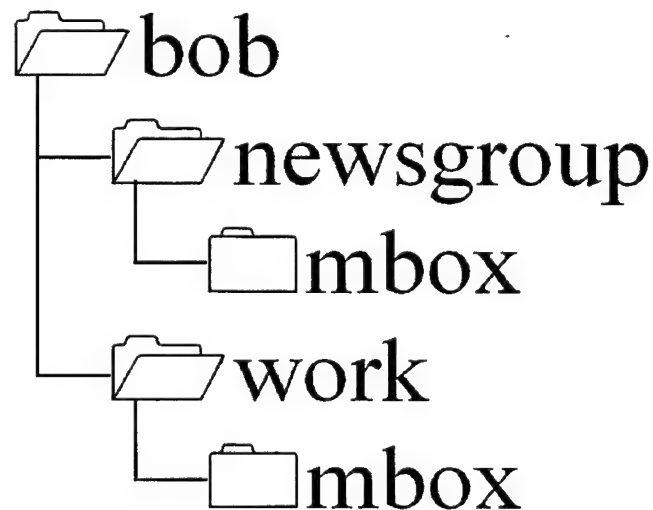
Figure 8. Sample Mailbox Hierarchy View From a Mail User Agent (MUA).

The mail folder metaphor is often used to describe a mailbox hierarchy. On the XTS-300, a mail folder corresponds to a directory. A mail folder can hold messages as well as hold other mail folders. IMAP does not have a structure that can contain both messages and other mailboxes. An IMAP mailbox only contains messages. IMAP mailboxes are either "selectable" or "not selectable". Only selectable mailboxes can be opened to be read. Mailboxes that are not selectable indicate a user does not have the permissions to read a named directory or file. Another way IMAP describes its mailboxes is the "inferiors" or "no inferiors" attribute. A mailbox that has inferiors is a directory, while a mailbox that has no inferiors is either a not selectable directory or a file. The IMAP literature normally only refers to mailboxes, but when a mail folder is mentioned it corresponds to an IMAP selectable mailbox with inferiors, or an XTS-300 directory. The difference is that IMAP mailboxes cannot support a nesting effect that mail folders can, i.e., a mailbox cannot contain both directories and files. IMAP forces all mailboxes that contain messages to have no inferiors.

IMAP4rev1 was designed for single level systems. The IMAP server and protocol do not support MAC and they are not multilevel "aware". As a result, the configuration of the mailbox hierarchy on the XTS-300 file system will determine how well the protocol performs in a multilevel environment.

B.   RESEARCH FOCUS AND APPROACH

1.   Trusted Mail Server Requirements

Bradley Eads' 1999 thesis [8] enumerated 10 requirements for a trusted mail server. These requirements are:

1. A user operating at a particular MAC level should be able to read all mail dominated by that MAC level (subject to DAC constraints).

2. A user operating at a particular MAC level should only be able to append to and send mail at that MAC level.

3. All mail read at a level should be marked as read at that level

4. Mail attachments should be supported at the client using COTS client mail software.

5. The server application must be constrained to the untrusted environment of the system on which it is operating.

6. It must provide for three levels of sensitivity and adhere to the TCSEC computer security requirements for open security environments.

7. It must store incoming e-mail messages at different MAC levels as is appropriate to their label.

8. It must allow the user to add information to messages on the server and store draft messages on the server to simplify their management.

9. It must be as unobtrusive to the user as possible.

10. The ability to mark all messages that are read, even if they are at a lower access class.[8]

Eads successfully demonstrated that an IMAP protocol server operating on the XTS-300 meets the requirements for 2, 3, 5, 6, 7, and 8. This thesis attempts to

40

determine what mailbox hierarchy is best suited to establish requirement number 1, and to develop the administration tool to implement this hierarchy on the XTS-300.

2. Comparing Mailbox Structures

The mailbox hierarchy utilized by Eads is comprised of the classification label followed by the user name. The mailbox names look like this:

'/usr2/mail/unclass/username/mbox',

'/usr2/mail/conf/username/mbox',

'/usr2/mail/secret/username/mbox', and

'/usr2/mail/topsecret/username/mbox'.

The '/usr2/mail' portion of the hierarchy is at the root, therefore it's MAC level must be at the lowest sensitivity and highest integrity level of the hierarchy. The major advantage of this mailbox hierarchy is that the system administrator only has to create four directories on the XTS-300 file system, one at each MAC level corresponding to the MAC label alias used in the mailbox name. Once these four directories are created, the IMAP server is able to complete the mailbox hierarchy at the appropriate MAC level in the hierarchy, one user at a time. This is done by issuing the 'CREATE' command from the server, or from a MUA.

The structure described above is possible because the IMAP server knows the user name and knows the home directory. In this case, the IMAP home was hard coded based on the current MAC level of the subject and the user name. Once the CREATE

41

command is issued for the mailbox the server checks to determine if the mailbox already exists. If the user is logged in at secret, the server instance acting on behalf of Bob knows that the correct home is '/usr2/mail/secret/bob' and looks for '/mbox' there. At this point the mailbox does not exist, only a portion of the home path: '/user2/mail/secret'. Since the mailbox does not exist the server creates it and fills in any directories needed to complete the path. Figure 9 shows the mailbox hierarchy after the CREATE command.
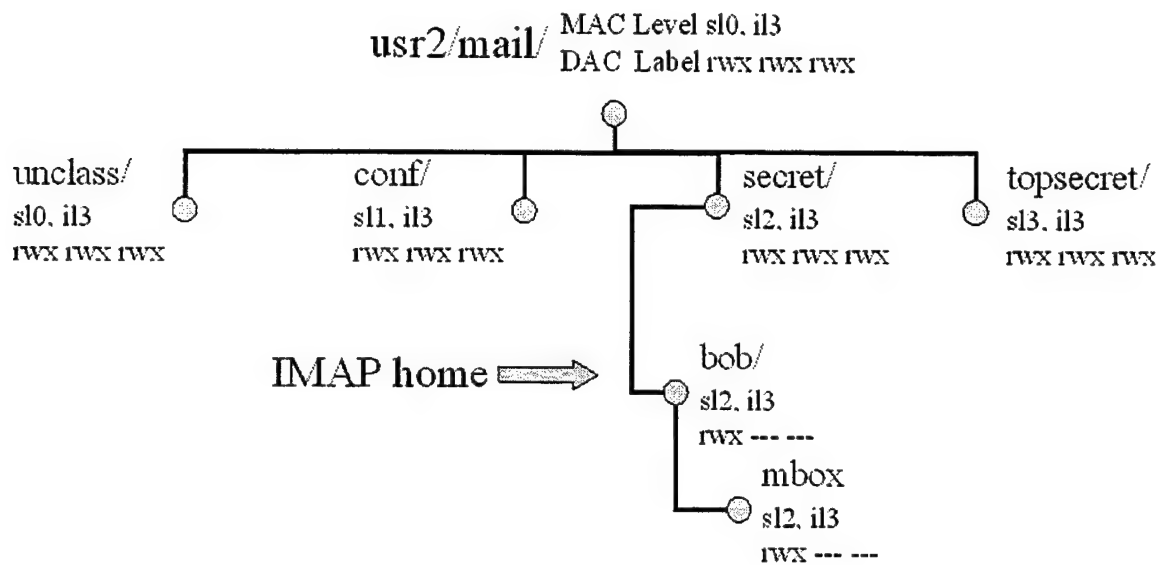
Figure 9. MAC-label-first Hierarchy After the IMAP CREATE Command.

Since neither IMAP nor any MUA's are trusted, each mailbox has to be created one at a time, at the level of the current session. However, a server in this configuration would allow mailboxes at the appropriate access classes to be created on a per user basis as needed. The administrator would only need to create the root MAC Label directories.

42

A major problem exists with this configuration in that it does not allow a read down into the lower MAC level mailboxes. IMAP4rev1 launches its search for mailboxes from its "home" directory. The server is able to visit all of the sub-directories of the IMAP home looking for mailboxes it is able to read. The default IMAP home is the user name directory. With the home set at '/usr2/mail/MAC_label/username', the server can only return a single level mailbox corresponding to the current MAC level of the user. This fails the requirement for the user to be able to read all the mail dominated by his current MAC level. Figure 10 shows the view from a MUA when user Bob tries to read his mail at secret.
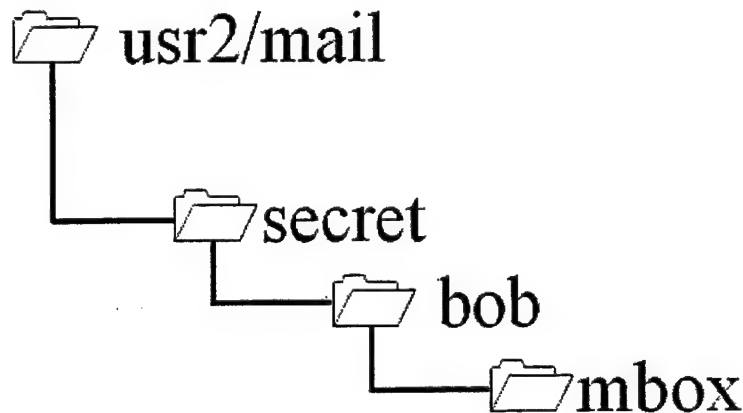
Figure 10. MAC-label-first Mailbox Hierarchy With No Read Down.

To correct this problem and enable the read down capability, the IMAP home should be at the root of the mailbox directory for a particular user. The root would be the directory with the lowest sensitivity level and highest integrity level that is common to all of the user's mailboxes. In the NPS MLS LAN implementation this corresponds to the directory '/usr2/mail'. This will allow the server to visit all of the sub-directories of the

43

IMAP home. This modification to the IMAP home enabled the read down capability, and in doing so created a separate problem. The problem is that the server now can see all of the sub-directories of the home directory, including the sub-directories of other users. For example, Bob is logged in at secret, and his home is /usr2/mail. When he checks his mail, the IMAP server not only returns:

'/usr2/mail/secret/bob/mbox',

'/usr2/mail/conf/bob/mbox',

'/usr2/mail/unclass/bob/mbox',

but also,

'/usr2/mail/topsecret',

'/usr2/mail/secret/mary',

'/usr2/mail/conf/mary', and

'/usr2/mail/unclass/mary'.

Figure 11 is a representation of how this hierarchy would look given the ability to read down.
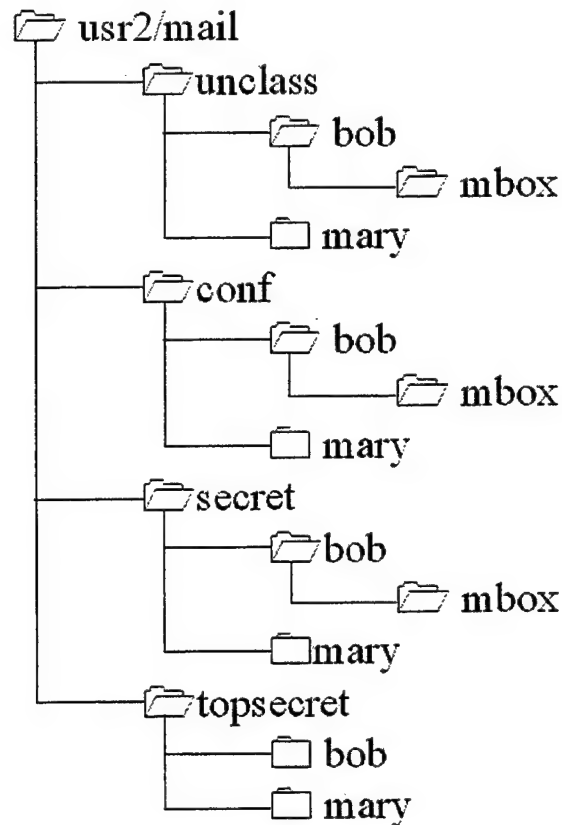
Figure 11. MAC-label-first Hierarchy With Read Down.

With a common mailbox root, it is possible for Bob to see what mailboxes exist for all the users at the secret level and below. This allows Bob to know who has access to secret information. This may violate some organizational policies such as "need-to-know". If Mary has her DAC policies set correctly, the XTS-300 would prevent Bob from reading Mary's mail. However, if Mary does not have her DAC permissions configured correctly, it is possible that Bob will be able to read her mail.

In addition, if Bob has many mailboxes, corresponding to each MAC level he is authorized to view. The server will return all of his mailboxes, only a fraction of which may be readable at the current session level. From a usability standpoint, the MUA will

45

be cluttered with mailboxes that the user cannot access. This may lead to user dissatisfaction. From a runtime standpoint, the server is wasting time looking for and returning mailboxes that the user cannot access.

Another problem with this hierarchy is that the organization is depending on IMAP4rev1 to create mailboxes at each level. IMAP4rev1 as it is configured sets the DAC policies on the mailboxes to be user only, non-sharable. However the IMAP code is stored in the untrusted domain and can be changed. This would allow the DAC policies to be set by an untrusted process. In addition, not all clients support the DAC commands that are supported by IMAP4rev1. Using an un-supportive client would mean that Mary would not only not know that Bob was reading her mail, but she would not be able to correct the problem from her client if she did. Besides having security drawbacks, this hierarchy representation can also be disorderly and user un-friendly. Every time a user checks his mail, he would see several mailboxes that do not belong to him.

Another way to solve the read-down problem is to reverse the user name and MAC level in the hierarchy. The user-name-first hierarchy will look like this:

/usr2/mail/username/unclassified,

/usr2/mail/username/classified,

/usr2/mail/username/secret, and

/usr2/mail/username/topsecret.

The /usr2/mail/username portion of the hierarchy serves as the mail root, therefore its MAC level must be at the lowest sensitivity and highest integrity level of the

hierarchy. The IMAP home would be this root directory. The IMAP LIST command would return only the selectable mailboxes that are subdirectories of the root. For example, Bob is logged in at secret and his IMAP home is /usr2/mail/bob. He checks his mail and the IMAP server would return:

/usr2/mail/bob/topsecret,

/usr2/mail/bob/secret/mbox,

/usr2/mail/bob/conf/mbox, and

/usr2/mail/bob/unclass/mbox.

Figure 12 is a representation of how the user-name-first hierarchy would look at a mail client.
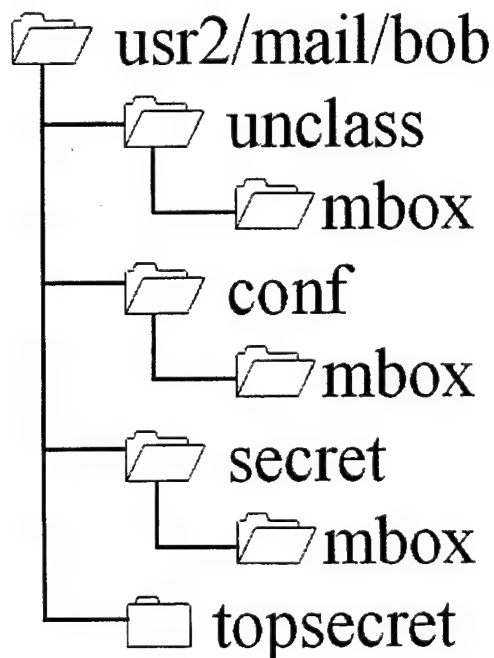
Figure 12. User-name-first Hierarchy.

47

Not only is the IMAP server able to read down in this configuration, but, by applying the DAC mechanism first in the hierarchy, Bob cannot read Mary's mailboxes. The directory /usr2/mail is owned by the administrator and readable by everyone in both hierarchies. In the MAC-label-first hierarchy, the '/MAC_label' directory is also owned by the administrator and readable by everyone. As a result, all of the MAC label subdirectories are readable by the server as it searches for a mailbox. It is not until the server visits the subdirectory /usr2/mail/MAC_label/mary that a restrictive DAC is placed on Bob's ability to traverse the hierarchy. In the user-name-first hierarchy, the IMAP home directory is '/username' and it is owned by that user. As a result, DAC is enforced at the first node the server visits.

The major problem with this hierarchy is that it requires an inordinate amount of work by a system administrator. Instead of creating four mail directories from which the IMAP server would automatically complete the mailbox hierarchy, the administrator would have to create the entire mailbox hierarchy for each user. In a small organization, creating this user-name-first hierarchy would be feasible using only the trusted commands already available on the XTS-300. In a large organization, however, the complex task of creating each mailbox at the appropriate MAC level necessitates an administrative tool. A mailbox administration tool must be a trusted process that is able to create user mailboxes at the requisite levels of the hierarchy with the proper MAC and DAC policies. With this tool, a secure multilevel mailbox hierarchy can be created and

48

managed with a negligible amount of hand crafted administration. Specifications for this tool are outlined in Appendix C and the source code is in Appendix D.

Another security problem exists within the XTS-300 file system. A parent directory is able to know about all of its sub-directories, regardless of the MAC level of the sub-directory. This was demonstrated in the earlier examples for both the MAC-label-first and user-name-first hierarchies. Even though Bob was logged in at secret, he was able to see he had a mailbox at topsecret. Sub-directories must be created at the same MAC level as the parent, otherwise the MAC sensitivity and integrity rules would be bypassed. In addition, the parent stores the names, segment ID and file system ID of its sub-directories when they are created. If a sub-directories' MAC label is upgraded through a trusted command, the sub-directory can only be read by a subject whose current MAC level dominates its label. The parent is no longer at the same MAC level as the sub-directory, but it does, however have a pointer consisting of its name, segment ID and file system ID. A person at the MAC level of the parent can see the name of the subdirectory but it cannot read it.

One way around this is to name the mailboxes so that a human reader cannot infer the level of the mailbox from the name. A 'topsecret' mailbox for Bob can be called "X" so that when she is logged in at 'secret', all she can see is the existence of a mailbox "X" that she cannot access. When Mary logs in at 'topsecret' she will be able to see "X" as well as the sub-directories of "X" which contain the "human readable" mailbox named "topsecret". The problem with this mailbox-naming scheme is that it can become

49

cluttered at the client with extra levels of the hierarchy needed to hide the name of the mailbox. Moreover, the user logged in at the lower level can still see that the higher level mailbox exists, he just cannot see the "human readable" name. This hierarchy is depicted in Figure 13.

```
📁 usr2/mail/bob
   ├── 📁 W
   │       └── 📁 unclass
   │               └── 📁 mbox
   ├── 📁 Q
   │       └── 📁 conf
   │               └── 📁 mbox
   ├── 📁 N
   │       └── 📁 secret
   │               └── 📁 mbox
   └── 📄 X
```
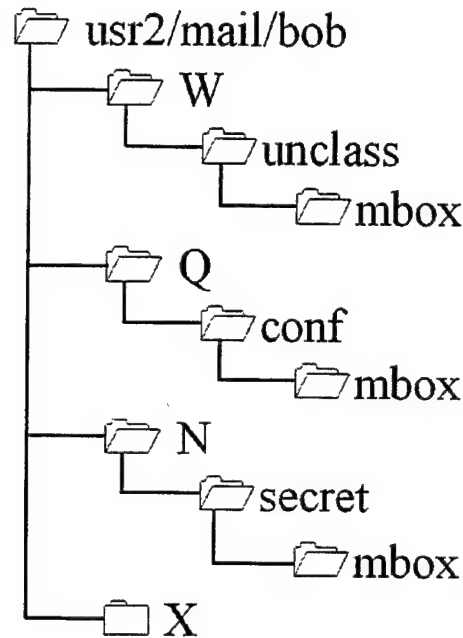
Figure 13. User-name-first Hierarchy With Obscured MAC Label.

IMAP has the functionality to filter out the extraneous levels of the hierarchy. Some MUA's such as *PINE* utilize this functionality, which makes this type of naming scheme more user friendly. However, if not knowing the existence of the higher level mailboxes is part of the organizational security policy, this scheme is not reliable. The mechanism for filtering the mailboxes from the lower level users would exist at the untrusted client.

Another method for filtering the names of the higher level mailboxes from the lower level users is being investigated. The mechanism for listing the names of files in

the XTS-300 is part of the **stat** system call. If the operating system code can be modified to filter the higher level mailboxes in the trusted computing base, it would provide a persistently secure solution to the knowability problem.

3.   Empirical Study

A small experiment was conducted involving the MAC-label-first, and the user-name-first mailbox hierarchies in order to:

1) assess the performance of the of the mailbox hierarchies,

2) to assist in the specification of the mailbox administration tool, and

3) to lay the groundwork for future experiments of the IMAP server and mail clients.

The two hierarchies were built and accounts for two users, Bob and Mary, and one group named WorkGroup were created. The accounts were created using the XTS-300 trusted commands **ga_edit** to create WorkGroup, and **ua_edit** to create Bob and Mary. These commands are only available to a person logged in with *administrator* privileges. The commands provide a menu for creating an account and selecting user privileges. Table 1 lists the information provided to the **ua_edit** command in order to create accounts for a user named Bob and a user named Mary.

51

ua_edit query	User Account Information	User Account Information
user name	Bob	Mary
user number(ID)	23	24
default group	Other	WorkGroup
command processor	/bin/sh	/bin/sh
home directory	/usr2/bob	/usr2/mary
Downgrade	No	No
Upgrade	No	No
viewing optional when downgrading	No	No
change user password	Yes	Yes
Disconnect	No	No
kill, ikill	Yes	Yes
Run	Yes	Yes
set group	No	No
set level, change default level	Yes	Yes
Shutdown	No	No
unmark printed output	No	No
multiple logins	Yes	Yes
max sensitivity level and cat	Max	Max
max integrity level and cat	il3 ic0 ic1 ic2	il3 ic1 ic2 ic3
user default sensitivity level	Min	min
user default integrity level	il3	il3

Table 1. Responses to the Trusted **ua_edit** Command When Creating an Account.

Each of the two hierarchies consisted of a user name and MAC label alias corresponding to an access class. The user name was either the login name of the user or the default group name of a user. The MAC level consisted of a sensitivity level (sl) and integrity level (il). The XTS-300 provides a security map database that contains the input and output strings for each of the 16 sensitivity levels, 64 sensitivity categories, 8 integrity levels and 16 integrity categories. An administrator is allowed to edit the input names, display names, and aliases in the database through the trusted **sm_edit** command. The input name is limited to five characters while the display name can be represented by

as many as nineteen characters. The input and display names represent a single sensitivity level, sensitivity category, integrity level or integrity category. For example, sensitivity level 1 can have an input name of 'sl1' and a display name of 'confidential'. The integrity level 3 can have an input name of 'il3' and a display name of 'trusted'. The alias is also limited to five characters but it represents an entire MAC level. For example, the MAC level sl1 (sc1) il3 (ic2) can be represented by 'al1'. When a user is wants to go to the MAC level sl1 (sc1) il3 (ic2), all he has to do is type in 'al1' when prompted for the sensitivity and integrity level of the session. The **sm_edit**, security map editor, allows the administrator to do the following:

- Add an alias for a MAC level.

- Change the input name.

- Change the display name.

- Change an alias name, or change a MAC level for a particular alias.

- Show a particular the display name or alias.

- List all the input names and display names, and aliases.

The security markings in the security map database are used on all file and directory displays performed by the **fsm** command and all output sent to the printers. This database could be useful in the naming of mailboxes by utilizing the aliases or display names provided. Further discussion of this topic is provided in Appendix C.

During the experiment, the pairing of MAC level with sensitivity label name went as follows: 'unclass' = (sl0,il3) | 'conf' = (sl1,il3) | 'secret' = (sl2,il3) | 'topsecret' =

(sl3,il3). Note that because all the integrity levels were the same, the names of these levels reflect only the secrecy classes.

DAC Permissions read (r), write(w), and execute (x) for each node in both hierarchies were specified. Each set of permissions can be set for any of the three types of users, the owner of the file (u), the members of the group to which the owner belongs (g), and all other users (o). These permissions are represented as a string of nine characters:

<u>u</u>   <u>g</u>   <u>o</u>

rwx   rwx   rwx

The experiment was conducted using standard IMAP4rev1 commands from the IMAP Server instead of from a client. The IMAP RFC 2060 lists a series of commands that "should" be run by the client. Each client interprets the RFC 2060 differently, so there is no guarantee that one command will work for all clients. By issuing orders in the command line from the IMAP server, we are able to see how the mailbox hierarchy performs, while ignoring client idiosyncrasies

The first hierarchy was the one adopted by Eads using the MAC label alias followed by the user name in the hierarchy. Mary is a member of WorkGroup and Bob is not. The hierarchy is created utilizing the trusted **fsm** (file system manager) command on the XTS-300. The **fsm** command can be accessed by normal users to manipulate files within their session level or by administrator users who have privileges to modify files and are trusted to use options that bypass the security policies. The **fsm** command is

menu driven and provides various functions for displaying, modifying, or deleting file system objects [16]. In order to create the required hierarchy, the administrator needs to be logged in at (min max), il3(ic0-ic15)). This is required since an administrator will be creating files and upgrading their MAC labels through a trusted option of the **fsm** command. This option is trusted because it is trusted to bypass the security *- property.

The **fsm** command has options to make directories The following directories were made first:

'/usr2/mail/unclass',

'/usr2/mail/conf',

'/usr2/mail/secret', and

'/usr2/mail/topsecret'.

Then using the **fsm** command again, the trusted change option was used to upgrade the '/conf' file to (sl1, il3), '/secret' to (sl2, il3), and '/topsecret' to (sl3,il3). The upgrade and downgrade functionality of the **fsm** command is only available to users who are specifically authorized to have this privilege activated when their account is being created or to users who are logged in with administrator privilege. It is necessary to keep this privilege isolated to only administrators who have a need to manipulate the file system as part of their jobs because of the potential for misuse and policy violation.

The change option also has the capabilities to modify the XTS 300 ACL which is more granular than the standard UNIX ACL. Each XTS-300 object has an ACL which consists of:

```
file owner:                     r w x
up to 7 other users:            r w x
owning group:                   r w x, and
up to  7 other user groups:     r w x.
others:                         r w x.
```

The only files created were the four MAC label files and the WorkGroup mailbox. The IMAP server was expected to be able to create the rest of the hierarchy. This was tested by issuing the IMAP CREATE command for a mailbox at each session level. The completed mailbox hierarchy is shown in Figure 14. Selected IMAP commands were then issued against the hierarchy to test the mailbox structure performance against the protocol. The results of the test are shown in Table 2.

Figure 14. Compete MAC-label-first Hierarchy.

56

CREATE	
Expected results	IMAP4rev1will be able to create a mailbox for a user at the MAC level the user is logged in at (No write down).
Actual Results	IMAP4rev1 was able to create a mailbox at the subject's current MAC level. The entire hierarchy for a particular MAC level could be created with a single CREATE command because IMAP4rev1 automatically creates directories in a path name if they do not already exist.
DELETE	
Expected results	IMAP4rev1 will be able to delete mailboxes at the MAC level the user access class
Actual Results	IMAP4rev1 was able to delete mailboxes at the current MAC level.
LIST	
Expected results	IMAP4rev1 will list all child mailboxes of the IMAP home. This will include all of the users mailboxes, all of the user mboxes that the session dominates, the mailboxes of all of the group or other users that the session dominates and the group or other user mboxes that are readable through MAC DAC permissions.
Actual Results	The mailboxes for all users and groups that were dominated by the current users session were returned.
SUBSCRIBE	
Expected results	IMAP4rev1 will set the subscribe bit on the mailbox at the current session level only. (No write down)
Actual Results	IMAP4rev1 created a file named '.mailfile ' at every level. Users were able to subscribe to mailboxes that the current session dominated. However when other users log into the system they are able to see the '.mailfile'. A .mailfile is Not created for each user.
UNSUBSCRIBE	
Expected results	IMAP4rev1 will null the subscribe bit on the mailbox at the current session level only. (No write down)
Actual Results	A user is able to unsubscribe from any mailbox listed in the .mailfile file.
LSUB	
Expected results	IMAP4rev1 will return the mboxes the current user is subscribed to at levels the current session dominates
Actual Results	Returns the list of subscribed mailboxes at MAC levels dominated by the current session. Other users are able to access the .mailfile with the LSUB command.
SELECT	
Expected results	IMAP4rev1 will return the mailbox that is requested if the MAC & DAC permissions allow it.
Actual Results	IMAP4rev1 returned the designated mailbox if the MAC and DAC permissions allowed it.
COPY	
Expected results	IMAP4rev1 will be able to copy the contents of one mailbox to another at the same MAC level.
Actual Results	IMAP4rev1 was able to copy the contents of one mailbox to another at the same MAC level.

Table 2. Results of Running IMAP Commands Against the Security Label First Hierarchy.

The second hierarchy was constructed with the user name first followed by the MAC label alias. In this hierarchy, Mary is a member of WorkGroup and Bob is not. This is the hierarchy that requires the use of an administration tool. For the previous hierarchy only the four MAC level hierarchies were created using the **fsm** command. When creating this hierarchy the **fsm** command is used to create four mailboxes for each user. The resulting hierarchy is shown in Figure 15. Selected IMAP commands were then issued against the hierarchy to test the mailbox structure performance against the protocol. The results of the test are shown below in Table 3.
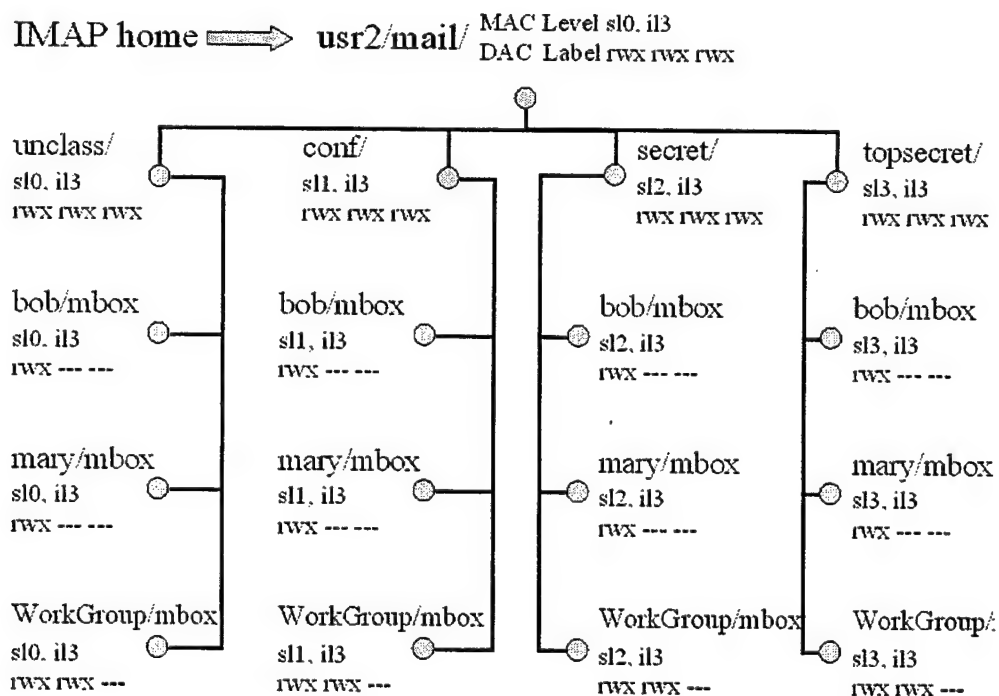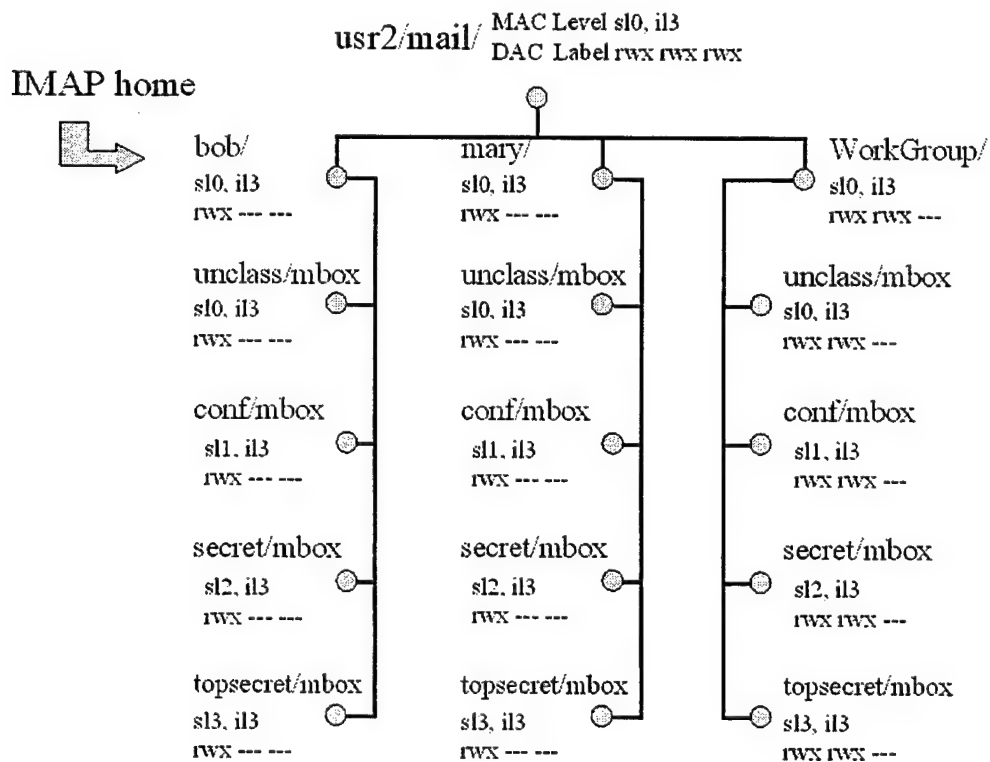

Figure 15. Complete User-name-first Hierarchy.

CREATE	
Expected results	IMAP4rev1 will be able to create a mailbox for a user at the MAC level the user is logged in at (No write down).
Actual Results	IMAP4rev1 was able to create a mailbox at the subject's current MAC level. The rest of the hierarchy was made through the trusted **fsm** command.
DELETE	
Expected results	IMAP4rev1 will be able to delete mailboxes at the MAC level the user is logged in at (No write down).
Actual Results	IMAP4rev1 was able to delete mailboxes at the current MAC level.
LIST	
Expected results	IMAP4rev1 will list all child mailboxes of the IMAP home. This will include all of the user mailboxes, all of the user mboxes that the session dominates, and the mboxes of all of the group or other user mailboxes that are readable through MAC DAC permissions.
Actual Results	A particular user mailbox that was dominated by the current user session was returned as well as the mailboxes that were owned by the group.
SUBSCRIBE	
Expected results	IMAP4rev1 will set the subscribe bit on the mailbox at the current session level only. (No write down)
Actual Results	IMAP4rev1 created a file named '.mailfile ' in the home directory. Users were able to subscribe to mailboxes that the current session dominated. A .mailfile is created for each user.
UNSUBSCRIBE	
Expected results	IMAP4rev1 will null the subscribe bit on the mailbox at the current session level only. (No write down)
Actual Results	A user is able to unsubscribe from any mailbox listed in the .mailfile file.
LSUB	
Expected results	IMAP4rev1 will return the mboxes the current user is subscribed to at levels the current session dominates
Actual Results	Returns the list of subscribed mailboxes at MAC levels dominated by the current session.
SELECT	
Expected results	IMAP4rev1 will return the mailbox that is requested if the MAC & DAC permissions allow it.
Actual Results	IMAP4rev1 returned the designated mailbox if the MAC and DAC permissions allowed it.
COPY	
Expected results	IMAP4rev1 will be able to copy the contents of one mailbox to another at the same MAC level.
Actual Results	IMAP4rev1 was able to copy the contents of one mailbox to another at the same MAC level.

Table 3. Results of Running IMAP Commands Against the User-name-first Hierarchy.

C. CONCLUSION

Based on the experiment a clear preference for the design of the mailbox hierarchy was established. The user-name-first hierarchy was chosen because it does not allow users to select each other's mailboxes, it is represented in a cleaner, more logical fashion by the client system, and most importantly it enables a read down capability. By selecting this hierarchy, a greater burden is placed on the administrator to create all the required mailboxes for each user. This burden can be reduced by designing a trusted process to create the mailboxes automatically. Appendix C explains the steps in designing the process and gives specifications for the **mailtool** code.

IV. TOOL EVALUATION

A. PROCEDURE

A prototype mailbox creation application named **mailtool** was produced and tested as a trusted command. Among other things, trusted commands allow a user to manipulate the MAC and DAC attributes for the current session, and to create, attach, and destroy process families at different MAC labels. To execute the command, the user must possess the appropriate capabilities. However, users of a trusted command may be able to use the tool, intentionally or unintentionally, to subvert other security processes [12]. All violations of MAC and DAC policy must be justified and kept to a minimum. Privileges may not be held longer than necessary. Each instance of a privilege must be verified to be necessary, and tested for errors that may cause damage.

To test the **mailtool,** two accounts were created, as before, and a mailbox hierarchy was created for each individual. The mailtool is an interactive tool that queries the administrator for inputs. At each input point the tool was tested to see if improper inputs would result in the tool proceeding to an unexpected state. Each request was tested to see if proper inputs were accepted and that the proper inputs had the expected results. The main function of the mail file administration tool is to allow administrators to easily create mailboxes for users. The e-mail administrator should be able to set up the mailboxes for each user and not contend with the underlying mail file structure. Several

criteria were used in testing the tool for accuracy, integrity and usability. These criteria include:

- Does the mail tool set proper DAC attributes on the mailbox hierarchy?

- Does the mail tool set the proper MAC attributes on the mailbox hierarchy?

- Does the mail tool properly constrain the MAC levels available for each user mailbox?

- Are clients able to access the mailboxes with the IMAP commands?

- Does the IMAP4rev1 server interpret the mailbox hierarchy appropriately? To test this, seven of the ten requirements for a trusted mail server listed in Chapter 3 as specified by Eads [8] were used. These criteria were selected because each requirement is a consequence of server and mailbox interaction. Specifically it is a representation of how the IMAP protocol functions within the TCB. These requirements are:

1. A user operating at a particular MAC level should be able to read all mail dominated by that MAC level (subject to DAC constraints).

2. A user operating at a particular MAC level should only be able to append to and send mail at that MAC level.

3. All mail read at a level should be marked as read at that level

4. It must provide for three levels of security and adhere to the TCSEC computer security requirements for open security environments.

62

5. It must store incoming e-mail messages at different MAC levels as is appropriate to their label.

6. It must allow the user to add information to messages on the server and store draft messages on the server to simplify their management.

7. It must allow a user to mark all messages that are read, even if they are at a lower access class. [8]

- Does the mail tool code support a mail file architecture that is not overly complex?

B.   FINDINGS

The mail tool essentially makes directories and files based on Administrator input. The input response to each query must be tested for proper type checking

?   Does the mail tool set proper DAC attributes on the mailbox hierarchy?

✓   YES. The first directory that the tool is allowed to make is the user name directory. The user name directory is created in the home mail directory that is represented by the string '/usr2/mail'. This directory is owned by an administrator with the DAC label rwx for owner, group and other. The user name directory is created with the effective and real user ID and user login name taken from the 'etc/passwd' file. The user login name is concatenated with the '/usr2/mail' directory string to produce a string '//usr2/mail/username'.   The DAC permissions for the user name directory are pre-defined as rwx for owner.   The

63

user name directory string is passed in the TCB gate call **create_directory**, and a user name directory is created with the proper permissions. Error checking occurs in the TCB gate call. All subsequent calls to **create_directory** are called with the same DAC permissions.

? Does the mail tool set the proper MAC attributes on the mailbox hierarchy?

✓ YES. The MAC level directory is created in user name directory. The MAC label for the MAC level is retrieved from the alias database and concatenated to the user name directory to form the string '/usr2/mail/username/MAC_label'. The MAC level of the mail directory and user name directory is sl0, il3. This represents the lowest sensitivity level and highest integrity level of the hierarchy. When the MAC level directory is created, it has the same MAC label as the parent directory. This directory is then upgraded to the desired MAC level through the TCB gate call **set_fd_access**. The TCB call takes a file descriptor and the desired MAC access class and sets the MAC level of the directory. Error checking takes place in the TCB gate call.

? Does the mail tool properly constrain the MAC levels available for each user mailbox?

✘ NO. This functionality is still being implemented

? Are clients able to access the mailboxes with the IMAP commands??

✓ YES. Clients are able to receive and edit mail in their mailbox hierarchy.

? Does the mailbox hierarchy perform as expected with IMAP4rev1?

64

1. A user operating at a particular MAC level should be able to read all mail dominated by that MAC level (subject to DAC constraints).

✓ YES. A user is able to access all of the mailboxes that the current session dominates.

2. A user operating at a particular MAC level should only be able to append to and send mail at that MAC level.

✓ YES. Current MLS LAN configuration prevents sending mail to users operating on remote high assurance servers, but the system does allow mail to be appended and sent to other users on the same local server.

3. All mail read at a level should be marked as read at that level

✓ YES. All mail in read within the current session MAC level is marked as read.

4. It must provide for three levels of sensitivity and adhere to the TCSEC computer security requirements for open security environments.

✓ YES. The XTS-300 supports 16 sensitivity levels plus 64 sensitivity compartments. Users are able to receive mail at all of these levels.

5. It must store incoming e-mail messages at different MAC levels as is appropriate to their label.

✓ YES. Incoming mail is stored on the spool and written into the appropriate location on the mailbox hierarchy.

6. It must allow the user to add information to messages on the server and store draft messages on the server to simplify their management.

65

✔ YES. Users are allowed to edit messages at the current session level and store them on the server.

7. The ability to mark all messages that are read, even if they are at a lower access class.

✘ NO. This functionality does not exist. A trusted process would have to be spawned each time a message is read. The process would require with the SIMPLE_SECURITY exempt privilege. Each instance of this process would trigger an auditable event. Allowing this process would possibly open up a window for a Trojan Horse program as high access classes would be able to write information to lower access classes.

? Does the mail tool code support a mail file architecture that is not overly complex?

✔ YES. The mail tool is a trusted command that allows an administrator to create a mailbox hierarchy for an authorized user. The Secure Server invokes the mail tool program to process this command. The program executes at the minimum sensitivity and administrator integrity level MAC Level (sl0, il3(ic0, ic2, ic3, ic4, ic5, ic6, ic7, ic8, ic9, ic10, ic11, ic12, ic13, ic14, ic15)). This ensures only users with administrator privilege may run the tool. If the program were allowed to execute at lower MAC levels, users and operators would be able to access execute the trusted process. This would violate the separation of privilege principle. The program has the following privileges:

- SIMPLE_INTEGRITY_EXEMPT: Allows the mail tool to bypass the simple integrity policy, i.e., it can read objects at an integrity label dominated by the mail tool integrity label. The mail tool is run at the administrator level, which has the highest integrity possible. The mail tool is required to read the '/etc/passwd' file which is at integrity level 3 (il3). The '/etc/passwd' file is a database of user information that is normally used in the CASS environment. The '/etc/passwd' file has seven fields, two of which are read by the mail tool **Get_User_ID** function. These fields are the user login name and the user ID. User ID and User names are required to make mailboxes for the appropriate user with the appropriate DAC privileges. The privilege is activated before the **Get_User_ID** call and is disabled immediately after the user name and ID are returned.

- SET_OWNER_GROUP: This allows the mail tool to set the real and effective owner of the mail tool process to that of the user whose mailboxes are being created. Essentially the mail tool will be running as that user. This privilege enables the tool to create directories and files as the user, thus automatically setting the DAC policies for the files as they are made. This privilege is called before the TCB gate call to **set_user_group** and disabled after the real and effective process user ID is changed.

- UPGRADE_LEVEL: Allows the mail tool to upgrade the MAC label on a directory structure. This privilege is required to create the multilevel mailbox

67

hierarchy that is desired. All files and directories are initially created with the MAC label of the root. After the directory has been created the MAC label is upgraded. A directory with a higher MAC label is required so that incoming and outgoing mail at higher classification levels can be created, stored, and edited at the proper level. This privilege is activated just before the mail tool Make_MAC_Directory procedure is called, and the original privileges are restored immediately after the call.

V. CONCLUSIONS AND RECOMMENDATIONS

A.   DISCUSSION

Electronic mail provides a convenient medium through which DoD personnel can perform various job functions efficiently. From memorandums to graphics, sound and video attachments, e-mail can transport a variety of electronic information. Electronic mail is even more versatile with the implementation of the IMAP protocol. Users can access their message stores from virtually anywhere. With increased accessibility comes an additional threat of information compromise through eavesdropping, viruses, and Trojan Horses.

A COTS mail server hosted on a high assurance TCB helps resolve some of these threats. The security kernel mediates all accesses to information stored on the server. MAC and DAC controls are placed on every attempt to access information. This creates a high assurance environment that is resistant to penetration and able to isolate breaches of security through hardware and software protection mechanisms.

Administration of the high assurance server is tedious due to the constraints placed on the file system. When setting up a mail hierarchy, for example, mailboxes need to pre-exist for each user at each classification level authorized for that user. To make these mailboxes the administrator needs to create directory structures and change their MAC and DAC levels to match the desired mail hierarchy. The trusted process developed in this thesis automatically creates the mailbox hierarchy for any system user. The

mailtool allows administrators to easily set up mailboxes for each user in any MAC Level that the user is authorized to access. The tool assists in the management of XTS-300 file structure and enables account administration for multiple LAN users at multiple security levels.

B.   RECOMMENDATIONS

Although the Group mailboxes have been created with the **fsm** command, and are accessible by members of the group, the **mailtool** does not automatically create them as it does individual accounts. Increased functionality needs to go into the tool to automatically create the group mailboxes. Two portions of the code need to be modified to make this possible. The first modification is to the TCB gate call to **set_user_group**. This call allows the calling process to set the owning group of the object. Currently only the owning user is being passed into this function and the owning group is left unchanged. The tool would need to get the desired owning group from a list of groups and then create the group-shared directory. This directory would have the group listed in its ACL. Next the process would have to change the DAC label to allow rwx to the owning group. After this is done, the object created will be able to be accessed by any member of the group.

C.   FUTURE WORK

Two specific problems need to be addressed to make the mailbox hierarchy more user friendly. The first problem was addressed earlier and has not been solved. A desired

function in this mail server implementation is the ability to mark all messages that are read, even if they are at a lower access. Allowing this process would possibly open up a window for a Trojan Horse program because high access classes would be able to write information to lower access classes. A trusted process would need to be implemented that interacts with the XTS-300 file system and IMAP. The main purpose for this type of function is for user friendliness. If a user reads all of his mail in a session labeled secret, he should not have to go back in at unclassified to delete all of the mail he just read.

Another problem is security policy related and it has to do with how the file system is presented to the user. Control of creation and deletion of hierarchical structured objects, such as those in the file system, are based on the ability to write to the directory containing the object. This means that a subdirectory is created at the same MAC level as the root directory. The root directory retains the file information (name, Segment ID, FS ID) for the subdirectories even if they are later upgraded [12]. This enables a user at the root level to be able to see all of the information for all of the subdirectories. In the context of the mail hierarchy, this means that a person logged in at unclass will be able to see that he has a mailbox at the secret level. The secret mailbox will not be accessible. If the person tries to read or select the secret mailbox, the security kernel will send a fault message that the file or directory does not exist. The file does, however, exist, and because the file name is stored at the root level, anyone with access to the root can see that it exists.

This may be a security violation depending on the organizational security policy. It is not a violation of TCSEC criteria for class B3 systems, therefore this XTS-330 file system implementation will likely remain. There is however a possibility that the "existence" of higher level subdirectories can be hidden by modifying the OSS domain library **stat** system call. The **stat** system call obtains information about a file system object. The specifications for the **stat** system call state that rwx permissions are not required for a named file to return all of the information on that file [15]. However the code may be able to be modifies so that a filter is used before information on a named file is returned. The filter should return only file information on objects that are dominated by the current access class of the subject. If the filter works a person whose access class is at secret would not be able to see the mailboxes stored at topsecret.

A problem addressed earlier that has not been solved is the ability to mark all messages that are read, even if they are at a lower access. Allowing this process would possibly open up a window for a Trojan Horse program because high access classes would be able to write information to lower access classes. A trusted process would need to be implemented that interacts with the XTS-300 file system and IMAP. The main purpose for this type of function is for user friendliness. If a user reads all of his mail in a session labeled secret, he should not have to go back in at unclassified to delete all of the mail he just read.

APPENDIX A. GLOSSARY

Access – An interaction between a subject and an object that results in the flow of information from one to the other [21].

Access Control- A method of limiting access to objects to only authorized subjects access [21].

Access Class- A combination of sensitivity label and integrity label. Also referred to as a MAC label [12].

Compartment- A class of information that has need-to-know controls beyond those normally provided for access to Confidential, Secret or Top Secret information [21].

Display Name- The human readable name of a particular element of a security or integrity label. The XTS-300 has a security map database where the display names of all sensitivity levels, sensitivity categories, integrity levels and integrity categories are defined.

Dominate- Given two MAC labels, the first is considered to "dominate" the second if the hierarchical level of the first is greater than or equal to that of the second, and if the category set of the first is a superset of the second. This comparison rule holds for both sensitivity and integrity labels. To obtain read access the sensitivity labels of the subject must dominate the object, for integrity labels the object dominates the subject [12].

High Assurance- A measure of confidence that the security features and architecture of a system accurately mediates and enforces security [21]. With respect to the TCSEC, class B2 and above are considered high assurance.

Human Readable Label- A printable label name associated with exported sensitivity labels [21].

Integrity Label- Comprised of one of 8 hierarchical integrity levels (il) and from 0 to 16 nonhierarchical integrity categories (ic) [12].

MAC Label- A combination of a sensitivity label and an integrity label [12].

MAC Label Alias- A human readable name that describes a MAC Label. This alias is used in **mailtool** to name mailboxes.

Mailbox- A formatted container that holds e-mail messages [1].

Multilevel security- A class of system containing information with different sensitivities that simultaneously permits access by users with different security clearances and needs-to-know, but prevents users from obtaining access to information for which they lack authorization [21].

Object- A passive entity that contains information such as a directory or a file [21].

Privilege- A protection mechanism that provides a controlled mechanism whereby a process operating on behalf of a user can be authorized to bypass the system security policy in a selected fashion [12].

Sensitivity Label- Comprised of one of 16 hierarchical security levels (sl) and from 0 to 64 nonhierarchical security categories (ic) [12].

Subject – A process, or a program in execution [21].

APPENDIX B. STOP 4.4.2 PRIVILEGES

This Appendix lists privileges used by the XTS-300 TCB [11] [12]. A privilege is a controlled mechanism whereby a process operating on behalf of a user can be authorized to bypass the system security policy in a selected fashion [11]. Each process has a maximum set of privileges it may possess at any one time. Only trusted processes have privilege because with privilege, a process can bypass specified security policy.

- ☐ MODIFY_PRIVILEGE- Allows a process to modify its maximum privilege set.

- ☐ SET_LEVEL- Allows a process to change the MAC label on an object.

- ☐ UPGRADE_LEVEL- Allows a process to upgrade the MAC label on an object.

- ☐ SET_DISCRETIONARY_ACCESS- Allows a process to change the access control list of an object if it is not the owner of the object.

- ☐ SET_OWNER_GROUP- Allows a process to change the access control list of an object, the other mode bits (i.e., setuid/setgid) of an object, or the owning user/group of the object, even if it is not the owner of the object.

- ☐ SET_PROCESS_ATTRIBUTES- Allows a process to set its clearance label and process family.

- ☐ SET_SUBTYPE_ACCESS- Allows a process to change the current subtype of an object.

- ☐ TERMINAL_LOCK- Allows a process to retain control of the terminal when a secure attention key is pressed.

- ☐ DEVICE_CONTROL_EXEMPT- Allows a process to perform primitive hardware control function on a device.

- ☐ SIMPLE_SECURITY_EXEMPT- Allows a process to bypass the simple security property, i.e., it can read objects at a sensitivity label that dominated the sensitivity label of the process.

- [ ] SECURITY_STAR_PROPERTY_EXEMPT- Allows a process to bypass the security *-property, i.e., it can write objects with sensitivity labels dominated by the process's sensitivity label.

- [ ] SIMPLE_INTEGRITY_EXEMPT- Allows a process to bypass the simple integrity property, i.e., it can read objects at an integrity label dominated by the integrity label of the process.

- [ ] INTEGRITY_STAR_PROPERTY_EXEMPT- Allows a process to bypass the integrity *-property, i.e., it can write objects with integrity labels that dominated by the process's integrity label.

- [ ] DISCRETIONARY_ACCESS_EXEMPT- Allows a process to bypass the discretionary access and subtype policies.

- [ ] TRUSTED_PARENT_EXEMPT- Allows a process with an integrity level below operator to load trusted processes with privileges. When the privilege is present at the time of loading, the privilege bits of the new process are not zeroed when the creating process is untrusted. It is not currently used by stop 4.4.2. [11]

APPENDIX C. MAILTOOL DESIGN

A. REQUIREMENTS

In Chapter III the mailbox hierarchies described were created with a root directory at the MAC level sl0, il3. Subdirectories were initially created at that level and upgraded through the **fsm** command. The ability to use the **fsm** command to upgrade objects is only available to privileged users. Each user has several mailboxes corresponding to the various levels of classified information they are allowed to access. The system administrator needs a tool to help create mailbox hierarchies for the many users in the organization. This Appendix provides the specification for a trusted administration tool that will automatically create and upgrade the directories that make up the mailbox hierarchy. This tool will greatly reduce administrator workload and errors possible by manually using the **fsm** command. This hierarchy structure allows a user to login, receive and send mail at the current session level. The user is also able to "read down" the mailbox hierarchy to mailboxes at the lower levels.

B. CONCEPTUAL MODEL

Figure 16, is the conceptual model of the trusted process named **mailtool**. It indicates the major entities in the mailbox structure and the relationships that exist between them. The conceptual model is used it help identify interactions between entities. These interactions become tasks that are then developed into procedures. A procedure is then defined by specifications that are eventually written into a code.

77
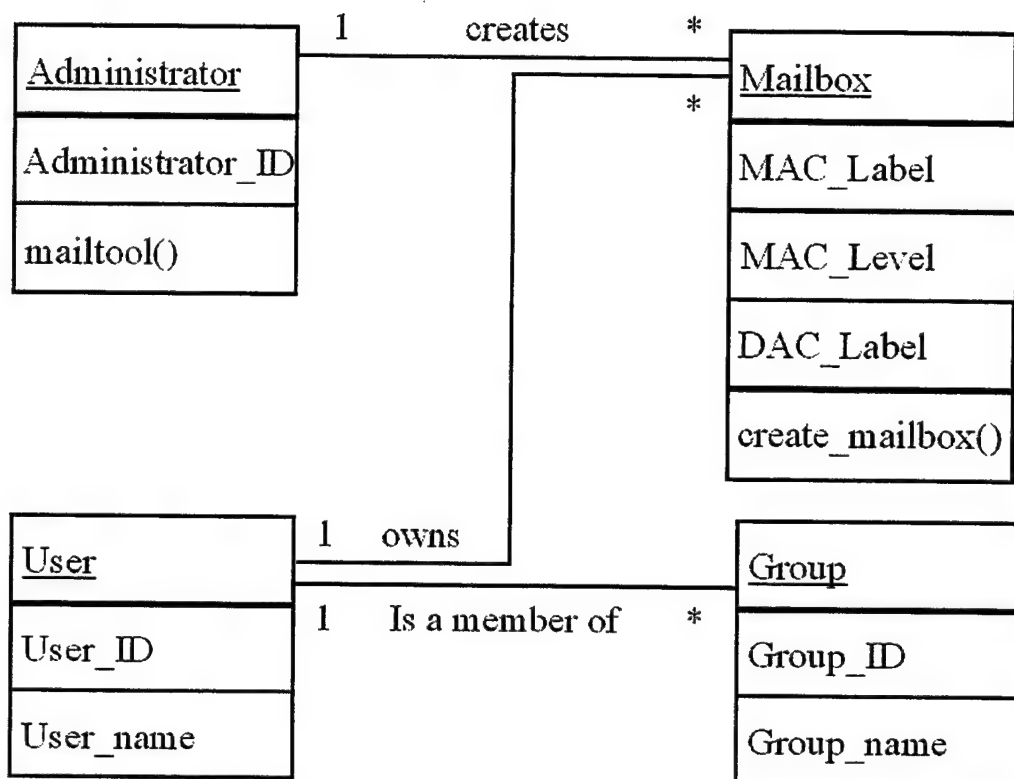
Figure 16. Conceptual Model of the Administrator Tool.

There are three distinct roles in the XTS-300 environment:

▯ System administrator

▯ Operator

▯ User [14]

Two of these roles are incorporated as entities into the conceptual model. The first

entity is the administrator. The administrator is responsible for implementing security

policies and procedures related to the TCB [14]. **Mailtool** is a trusted process because it

has the ability to change the DAC labels and upgrade the MAC labels of objects, therefore it must be used by someone occupying the administrator role. The mail administrator is the only user of the **mailtool** trusted process.

The mail administrator has as an attribute "Administrator ID", which defines the particular person occupying the role and describes what permissions that person is allowed to have. An attribute is identified in the conceptual model by placing it in a box below the entity. The Administrator ID is used to identify a person, or process acting on behalf of a person, to the system. The Administrator ID is used for both MAC and DAC access checks by the TCB throughout the **mailtool** process.

The administrator is the user of a process called **mailtool** that builds mailboxes. The conceptual model denotes this by placing a function called "mailtool()" under the administrator attributes. The relationship between the administrator and mailbox is the "creates" relationship, which is specified by a line connecting the two entities. The number '1' and symbol '*' on the line denote a "one-to-many" relationship, meaning the administrator may create as many mailboxes as needed.

The next entity, which is also an XTS 300 role, is the user. A user is not trusted and therefore cannot activate, deactivate, or bypass any security features of the system [14]. The user has the attributes "User ID" and "User Name". The User ID identifies the user to the system. It is used in setting DAC permissions on objects as well as various MAC and DAC access checks throughout the **mailtool**. The User ID is also used to match a user with a login name so that the mailboxes can be named properly. The User

Name attribute is the login name of the user and it is used in the path name of the mailbox.

A one-to-many relationship exists between the user and the mailbox entity. The relationship is an "owns" relationship that specifies a particular user may be the owner of as many mailboxes as needed. The User ID is entered into the owner portion of a mailbox object ACL. A one-to-many relationship also exists between the user and the group entity. The relationship is a "is a member of" relationship that specifies a user may belong to as many groups as needed. The actual maximum number of groups is 1023 [14], however for practical purposes the one-to-many relationship is used. The group entity has attributes "Group ID" and "Group name". These attributes are used by the TCB for access control.

The final entity in the conceptual model is the mailbox. The mailbox has the "MAC label", "MAC level", and "DAC label" attributes. The MAC label is a human readable string name that is used in the mailbox pathname. The MAC level is a combination of the sensitivity and integrity level of the mailbox object, while the DAC label is the DAC ACL associated with the mailbox object. The relationships between the mailbox and other entities have already been described.

A task analysis is a narrative review of how the **mailtool** is to be used. The task analysis is conducted to determine what kinds of functions need to be designed within the process. The task analysis of the **mailtool** is as follows. The mail administrator will login

80

and invoke the trusted path. At the prompt, the administrator will invoke the **mailtool** trusted procedure.

The **mailtool** will first ask the administrator for the User ID or User Name of the person whose mailbox is to be constructed. If the entry of the User ID or User Name is valid, the tool will proceed to the next task. If the User ID or User name is not valid, the administrator will have to enter either another ID, a name, or exit.

The tool will then ask the user for the path name of the mail home directory. This directory must be known to both the IMAP server and the **mailtool.** For this reason, a default mail home directory is given as a choice. If the Administrator wants to use a different home, it can be entered here.

The **mailtool** takes the mail home directory and the User name and creates a directory '/mailhome/username'. The '/mailhome' directory is at the root of the hierarchy so it has the lowest sensitivity level and highest integrity level of the entire hierarchy. The '~/username' directory has the lowest sensitivity level and highest integrity level available to the user as long as it is not lower/higher than the mailhome. Its DAC permission are set to 'rwx --- ---' with the user as the owner.

The next task is for the administrator to enter a list of MAC levels that is used to create mailboxes for a user. This is done by asking the administrator for the sensitivity level, including sensitivity categories, and the integrity level, including integrity categories. This information is combined into a MAC level that has all four elements.

81

The MAC level represents the level at which the user is authorized to receive mail. The list is created one MAC level at a time or by referencing a default list.

The MAC Levels are then converted into a human readable label that is used to create a directory. The human readable MAC label must be standard for every user in the organization. For that reason, an alias database is used. This feature is available using a mailbox alias database that is used by both **mailtool** and IMAP. As stated in Chapter III, the XTS-300 provides a similar alias database in the security map database. However, limitations on the name length of the alias make its use problematic for users. For example, each mailbox in our hierarchy occupies its own MAC level. A mailbox at sl3(sc1), il3 needs to be named differently than the mailbox at sl3,il3. These two mailboxes should not be named '/usr2/mail/bob/secret'.

If the alias capability provided by the security map database is used, this is not a problem because each mailbox will have a unique name. The problem would be in reading and interpreting the mailbox aliases since each is limited to only five characters. The mailbox at sl3, il3 might be named '/usr2/mail/bob/tst for "topsecret trusted" and the one at sl3(sc1), il3 would be named '/usr2/mail/bob/tsnt for "topsecret NATO trusted". This naming scheme would not support easily identifiable mailboxes at multiple MAC levels.

If the security map display name of each sensitivity and integrity level and category is utilized, a user would be able to identify exactly the access class of each mailbox. All that would have to be done is to concatenate the names of each level and

category and separate them by an underscore. Thus sl2,il3 becomes a mailbox '/usr2/mail/bob/secret_trusted'. The problem then becomes that the mailbox names will start to become too long if multiple categories are used. A mailbox at sc2(sc1,sc2), il3(ic1) is named '/usr2/mail/bob/secret_nato_chemical_trusted_electronicintel'. This name, although descriptive, may become too long for readability and comprehension.

After the human readable MAC label is obtained from the alias database for a particular MAC level, the next task is to create the MAC level directory. The MAC level directory is a subdirectory of the user name directory with the pathname '/mailhome/usrname/MAC_label'. The MAC level of the '/MAC_label' directory is then upgraded utilizing special permissions to match the desired MAC level.

After the MAC level directory is created, the 'mbox' file must be created. The mbox file used in the NPS MLS LAN is a version of the UNIX mailbox format. Essentially an mbox is an XTS-300 file that represents the IMAP mailbox structure. The newly created mailbox is represented by the pathname '/mailhome/username/ MAC_label/mbox'. Each new message to an IMAP mailbox is appended to the mbox file. The mbox file is created at the MAC level of the parent directory so its level does not have to be modified. After the mbox file is created for each MAC level on the list of mailboxes, the tool is finished with its task and exits. This task analysis is represented in Table 4.

Primary Task : Create MailBox Hierarchy
Subtask 1: Get_User_ID
Subtask 2: Get_Mail_Home
Subtask 3: Make_User_Directory
Subtask 4: Get_MAC_list
Subtask 5: Get_MAC_label
Subtask 6 : Make_MAC_Directory
Subtask 7: Create_mbox

Table 4. **Mailtool** Primary Task Analysis.

The Primary task "Create mailbox hierarchy" is encompassed in the file tool.c. This is the main program in **mailtool.** The file tool.c makes calls to procedures in the file toolbox.c. These procedures equate to the subtasks in the task analysis. The procedures are specified in the next section.

C.   TOOLBOX PROCEDURES

1.   Get_User_ID

a.   External Interface

int Get_User_ID(char* name, short int &id);

b.   Inputs

☐ None

c.   Outputs

☐ name

The login name of a valid user from the '/ect/passwd' file

☐ id

84

The User ID number from the '/ect/passwd' file

d. Processing

This function displays a menu and prompts the Administrator for input. The administrator is given a choice to either select a user from a list, enter the user name, or exit. If the administrator picks the list option, a list of users who have accounts is displayed and the administrator is prompted to pick one. If the entry is valid, a name and id is output and integer '1' is returned. If the administrator to enters a valid name at the prompt, the name and corresponding id is output and an integer '1' is returned. If an invalid name or id is input by the administrator the function displays an error message and prompts the administrator to select from the menu again. The final option for the administrator is exit. If exit is selected nothing is output and '-1' is returned. This Procedure uses the standard call **getpwuid** to get the user name from a given id and the standard call **get_user_number** to get a user id from a name [13].

e. Exceptions

If the user name and user id is not valid, the function displays an error message and prompts the administrator with the menu of selections. If the administrator exits the function, a valid input for user id and user name was not received.

2. Get_Mail_Home

a. External Interface

int Get_Mail_Home(char* home);

b. Inputs

☐ home

The string corresponding to an existing directory which acts as the default home.

c. Outputs

☐ home

The string corresponding to the home directory upon which the hierarchy will be built.

d. Processing

This function checks to see if the default home directory is valid. The default home directory is hard coded into the IMAP server. This default directory is specified in a file that is accessible to both **mailtool** and the IMAP server. If the default directory is valid then it is passed back as output.

e. Exceptions

If the default directory is not assessable to **mailtool,** an integer '-1' is returned. Otherwise an integer '1'.

3. Make_User_Directory

a. External Interface

int Make_User_Directory(const char* home, const char* name,
                                       char* directory);

b. Inputs

☐ home

The character string that corresponds to the mail home directory.

☐ name

The character string corresponding to the login name of the user whose mailboxes are being constructed.

c. Outputs

☐ directory

The character string consisting of the mail home directory concatenated with a character '/' and the string corresponding to the user name. The output directory string looks like '/mailhome/username/

d. Processing

This function takes mail home and the user name and makes the directory '/mailhome/username'. The TCB gate calls **create_directory**, and **open_fs** are used [13].

e. Exceptions

87

If the creation of the directory fails a standard error code from standard library error_code.h is output to the screen and '-1' is returned. Otherwise an integer '1' is returned.

4.   Get_MAC_List

a. External Interface

MAC_alias_list* Get_MAC_list(const short int id,
                                    int *num_of_entries)

b. Inputs

☐ id

The user id of the mailbox owner.

c. Outputs

☐ num_of_entries

The number of list entries.

☐ MAC_alias_list

A struct comprised of a human readable label and an MAC level.

d. Processing

The administrator is prompted with a list of options for entering the MAC levels for which mailboxes are going to be placed in the hierarchy.   The procedure uses a OSS domain library function **request_level** to add MAC levels to the MAC_alias_list [13]. It also uses the utility call **get_hrl_db_label**. Once all of the levels and labels have

been added to the list, the function exits outputting the list of structs and the number of entries on the list.

e. Exceptions

If there are invalid entries at any prompt, the function displays an error message and prompts the administrator with the menu of selections. If the administrator exits the function or there is an error, a null list is returned and a num_of_entries value of 0 is output.

5. Get_MAC_label

During the design of this function it was determined that an alias database needed to be created so that IMAP and **mailtool** could reference the mailbox names in a hierarchy. This database was created and a utility call **get_hrl_db_label** was defined. The **hrl_db.h** file is included in the toolbox.c file so that this function may be called. The function Get_MAC_label was not needed in **mailtool**.

6. Make_MAC_Directory

a. External Interface

Int Make_MAC_Directory(const char* user_directory,
                            const char* label,
                            access_ma mac,
                            char* MAC_directory);

b. Inputs

89

☐ user_directory

The character string that corresponds to the user directory '/mailhome/username'.

☐ label

The human readable MAC label provided by the hrl_db.

☐ Mac

The MAC level that corresponds to the label

c. Outputs

☐ MAC_directory

The directory string that is constructed by concatenating the user directory with a character '/' and the label. '/mailhome/username/MAC_label"

d. Processing

This function takes the user name directory '/mailhome/username' and the MAC label to make the directory '/mailhome/username/MAC_label'. The TCB gate calls **create_directory**, and **open_fs** [13] are used. After the directory is made the TCB gate call **get_fd_access** and **set_fd_access** is made to upgrade the directory to the appropriate MAC level. The process making this call must be privileged.

e. Exceptions

If the creation of the directory fails, or the directory is not assessable for upgrade, a standard error code from standard library error_code.h is output to the screen and an integer '-1' is returned.

7. **Create_mbox**

   a. External Interface

      int Create_mbox(access_ma* level);

   b. Inputs

      ☐ level

   c. Outputs

      None

   d. Processing

      This function calls an IMAP utility **mbxcreat** that creates a 'mbox' for a user at the current process MAC level. The IMAP utility is aware of the mailbox owner and used the level to put the mbox at the appropriate place in the pathname. Essentially an mbox is an XTS-300 file that represents the IMAP mailbox structure. The newly created mailbox is represented by the pathname '/mailhome/username/MAC_label/mbox'.

   e. Exceptions

      If the creation of the directory fails a standard error code from standard library error_code.h is output to the screen and '-1' is returned.

91

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. SOURCE CODE

This appendix contains the source code for **mailtool**. The implementations of some functions return void instead of int as prescribed by the specification.

```c
// File:    tool.c
// Author:  Richard K. Rossetti
// Date:    Sep 2000
// Purpose: Provide a tool to set up a mailbox
//          hierarchy for an IMAP server running
//          on  the WANG XTS-300 platform

#ifdef OSS_OPTION
#include <user_group.h>
#include <tcb_gates.h>
#endif
#include <stdio.h>
#include <sys/types.h>
#include <pwd.h>
#include <limits.h>
#include "toolbox.h"
#include "IMAP_xts.h"
#include "priv_util.h"

int main ( int argc, char **argv )
{
    char user_name[NAME_LENGTH];//user name
    short int user_id;// user ID
    char *mail_home= hard_code_base_dir; //mail home
    char user_dir_name[PATH_MAX];
    char MAC_dir_name[PATH_MAX];
    MAC_alias_list *MAC_level_list;
    access_ma MAC_level;
    char MAC_label[NAME_MAX];
    error_code the_err;
    ushort old_priv;
    int num_of_mas;

    printf("***    Welcome to the Mailbox Adminstrator Tool    ***\n");

        //allows program to violate simple integrity
    old_priv = enable_simple_integrity_priv();

        //gets the name and user id of the mailbox owner
    Get_User_ID(user_name, user_id);

    if( user_id != -1) // if UID not valid exit
    {
        printf("You selected user  %2d)",user_id);
        printf("  - %8s\n", user_name);

            //sets effective user to user_name
            //allows process to run as user_name
```

93

```c
        old_priv = enable_uid_priv();
        the_err = set_user_group(user_id,NO_GROUP_CHANGE,
                                 user_id,NO_GROUP_CHANGE);
        set_priv(old_priv);

        Get_Mail_Home(mail_home);//gets from a .h file

            //makes the ~/<user_name> directory
            //with user_name as the owner
        Make_User_Directory(mail_home, user_name,
                             user_dir_name);

            //gets a list of mailboxes to be created
        MAC_level_list = Get_MAC_list(user_id, &num_of_mas);

            //while loop runs through list one at a time
        int ix = 0;

        while(ix != num_of_mas)
        {

                //allows process to upgrade object
                //MAC levels
            old_priv = enable_upgrade_priv();

                //makes a single directory for a
                //MAC label and upgrades its
                //MAC level.  dir is owned
                //by user_name
            Make_MAC_Directory(user_dir_name,
                               MAC_level_list[ix].label,
                               &MAC_level_list[ix].level,
                               MAC_dir_name);

            set_priv(old_priv);

                //calls IMAP to format make mbox
            Create_mbox(&MAC_level_list[ix].level);

            ix++;   //increments pointer to get next
                    //mailbox on the list

        } // end while

    }//end if

    printf("\n***     Exiting mailtool     ***\n");

    return(0);

}// end main
```

```c
// File:    toolbox.c
// Author:  Richard K. Rossetti
// Date:    Sep 2000
// Purpose: Provide a tool to set up a mailbox
//          hierarchy on the XTS-300
#ifdef OSS_OPTION
#include <user_group.h>
#include <error_code.h>
#include <tcb_gates.h>
#endif
#include <access.h>
#include <terminal.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/fcntl.h>
#include <sys/types.h>
#include <pwd.h>
#include <stop/message.h>
#include "toolbox.h"
#include "hrl_db.h"


//----------------------------------------------
//-- PROCEDURE     : void Get_User_ID
//-- PURPOSE       : Gets the username and User ID
//--                     for a mailbox
//-- Pre           : Trusted process needs privilege to
//--                     "read down" to lower integrity
//--                     database
//-- Post          : User name and User Id initialized
//-- Exceptions    : Invalid input from the keyboard
//-- Return        : void
//-- Parameters    : char* name, short int &id
//-- ------------------------------------------

void Get_User_ID(char* name, short int &id)
{
   struct passwd *pw_ptr;
   char choice_str[10], *gets_res;
   id = 0;

      //begining menu
   printf("This tool allows you to make mailboxes for a user.\n"
          "You can start by entering the name of the user you \n"
          "wish to make the mailboxes for or select a user from \n"
          "a list of users.\n");

   while (id == 0){

      printf("Valid requests are:\n"
             "    name (enter the name of the user)\n"
             "    list (select user from a list)\n"
             "    exit (exit this command)\n");

      gets_res = gets(choice_str);  //get the keyboard entry;
```

95

```c
if (!strcmp(choice_str,"list"))//if the menu selection is list
{
    int first_col = TRUE;

    // itterates through pwd file and prints elements of
    //passwd struct in two collums
    while (pw_ptr = getpwent())
        {
            printf(" %2d) ", pw_ptr->pw_uid);
            printf(" %8s", pw_ptr->pw_name);

            switch (pw_ptr->pw_uid)
            {
                case 0:
                case 1:
                case 2:
                case 3:
                case 4:
                case 8:
                case 17:
                {
                    printf("  NOT SELECTABLE");
                    break;
                }
                default:
                {
                    printf("                 ");
                }
            }//end switch

            if (first_col)
            {
                printf("    ");
                first_col = FALSE;
            }
            else
            {
                printf("\n");
                first_col = TRUE;
            }// end if else

        }//end while

    if (!first_col)
    {
        printf("\n");
    }

    //gets the user ID from keyboard
    printf("\nEnter the number of the user"
            " you wish to create a mailbox for: \n");
    gets_res = gets(choice_str);
    id =  atoi(choice_str);//the number entered from the keyboard;

    pw_ptr = getpwuid(id);
```

96

```c
        if (pw_ptr != NULL)
        {

            switch (id)
            {
                case 0:
                case 1:
                case 2:
                case 3:
                case 4:
                case 8:
                case 17:
                {
                    printf("ERROR : The ID is not selectable\n");
                    id = 0;
                    break;
                }
                default:
                {
                    strcpy(name,pw_ptr->pw_name);
                }
            }
        }
        else
        {
            id = 0;
        }// end if

    }// end if

    else if(!strcmp(choice_str,"exit"))//if the menu selection is exit
    {
        id = -1;
    } //end else

    else   //get the name
    {

        id = get_user_number(choice_str);

        if (id != NULL_OWNER)
        {
            switch (id)
            {
                case 0:
                case 1:
                case 2:
                case 3:
                case 4:
                case 8:
                case 17:
                {
                    id = 0;
                    break;
                }
                default:
                {
```

```c
                        strcpy(name,choice_str);
                }
            }

        }//end if

        else
        {
            id = 0;
        }
     }// end else

    if (id == 0)
    {
      printf("You entry was not valid, try again\n");

    }//end if

  } //end while
}
//-------------END Get_User_ID------------------------


//------------------------------------------------------
//-- PROCEDURE    : void Get_Mail_Home
//-- PURPOSE      : Get the mail home
//-- Pre          : Default mail home defined
//-- Post         : mail home directory defined
//-- Exceptions   : Invalid input from the keyboard
//--                         string too long
//-- Return       : void
//-- Parameters   : char* home
//-- ------------------------------------------------

void Get_Mail_Home(char* home)
{
char choice_str[32], *gets_res;
short int value = 0;

 printf("The default mail home is %s\n",
          home);

    if(request_yes_no("do you want to use this as your default
directory?"))
    {
        printf("yes\n");
    }
    else
    {
        printf("no\n");
    }
}
//-------------END Get_Mail_Home------------------------


//------------------------------------------------------
```

```
//-- PROCEDURE    : void Make_User_Directory
//-- PURPOSE      : makes the directory /home/username
//-- Pre          : mail home defined, user defined
//-- Post         : a directory is created
//-- Exceptions   : directory already exists
//-- Return       : void
//-- Parameters   : const char* home, const char* name,
//--                      char* directory
//-- -------------------------------------------------

void Make_User_Directory(const char* home,
             const char* name, char* directory)
{
   error_code err;
   access_da dac ;
   dac.owner_perms =READ_WRITE_EXECUTE_MODE;
   dac.group_perms = 0;
   dac.other_perms =0;
   sprintf(directory, "%s/%s", home, name);

   printf("Making directory %s\n", directory);
   err = create_directory(directory,dac);//maked dir w/ rwx for owner

   if (err != NO_ERROR)
   {
      print_error("ERROR Creation of user directory failed ", err);
   }

}
//-------------END Make_User_Directory-------------------


//-------------------------------------------------------
//-- PROCEDURE    : void Get_MAC_list
//-- PURPOSE      : Gets a list of MAC labels and MAC
//--                      label aliases
//-- Pre          : user id defined
//-- Post         : A list of MAC labels and alias labels
//-- Exceptions   : Invalid user input
//-- Return       : MAC_alias_list struct
//-- Parameters   : short int id, *num_of_entries
//-- -------------------------------------------------

 MAC_alias_list*  Get_MAC_list(short int id,
                      int *num_of_entries)
{
   const char* alias[5] = {"unclass", "conf",
                       "secret", "topsecret",NULL};
   MAC_alias_list *list = NULL;// = malloc(4 *sizeof(MAC_alias_list));
   int index=0;
   *num_of_entries = 0;
   int keep_going = 1;
   int yes_label = 1;
   int add_to_list = 1;
   access_ma temp_ma;
   char* temp_label;
```

```c
        init_hrl_db_for_update();

        char choice_str[10], *gets_res;

        printf("\nYou may create mailboxes from a default list,\n"
                "you may create  them individually, or you may create\n"
                "them from a range of MAC levels.\n");

        while(keep_going == 1)
        {
            printf("Valid requests are :\n"
                    "    default (create mailboxes from a default list)\n"
                    "    single  (create individual mailboxes)\n"
                    "    range   (create mailboxes from a range of MAC
levels)\n"
                    "    display (display the mailboxes owned by the user)\n"
                    "    list    (show the list of mailboxes to be made)\n"
                    "    make    (make the mailboxes on the list)\n"
                    "    exit    (exit without making any mailboxes)\n");

            gets_res = gets(choice_str);

            if (!strcmp(choice_str,"default"))//if the menu selection is
default
            {
                printf("Making the default list\n");

                while(alias[index] != NULL)
                {
                    //printf("*** add of level %d\n",
                    //(int) &(list[*num_of_entries].level));

                    if (get_hrl_db_access_class(alias[index],
                        &temp_ma)==HRL_FOUND)
                    {
                        if (list == NULL)
                        {
                            list = malloc(sizeof(MAC_alias_list));
                        }
                        else
                        {
                            list = realloc(list,
((1+(*num_of_entries))*sizeof(MAC_alias_list)));
                        }//end if else

                        memcpy(&list[*num_of_entries].level,
                                &temp_ma, sizeof(access_ma));
                        strcpy(list[*num_of_entries].label, alias[index]);

                        printf("***MAC level sl: %d il: %d\n",
                            list[*num_of_entries].level.security_level,
                            list[*num_of_entries].level.integrity_level);

                        (*num_of_entries)++;

                    }
```

100

```c
            else
            {
                printf("ERROR: alias not found;    %s\n", alias[index]);

            }//endif

            index++;

        }//end while
    }//end if (default)

    else if (!strcmp(choice_str,"single"))//if the menu selection is
single
    {
        request_level("mailbox",&temp_ma,FALSE, TRUE);

        if (get_hrl_db_label(&temp_ma,
                        temp_label)==HRL_FOUND)
        {
            add_to_list = 1;
        }
        else
        {
            display_level("\na label has not been defined for ",
                            &temp_ma);
            yes_label =
                    request_yes_no("\nDo you want to define one?\n");
            if(yes_label)
            {
                display_level("enter the mailbox label for :",
                                &temp_ma);
                gets_res = gets(temp_label);
                add_hrl_db_entry(&temp_ma, temp_label);
            }
            else
            {
                add_to_list = -1;
            }//end if else
        }// end if else

        if(add_to_list)
        {
            printf("found\n");
            if (list == NULL)
            {
                list = malloc(sizeof(MAC_alias_list));
            }
            else
            {
                list = realloc(list,
                    ((1+(*num_of_entries))*sizeof(MAC_alias_list)));
            }//end if else

            memcpy(&list[*num_of_entries].level,
                    &temp_ma, sizeof(access_ma));
            strcpy(list[*num_of_entries].label, temp_label);
```

101

```c
          printf("***MAC level sl: %d il: %d\n",
              list[*num_of_entries].level.security_level,
              list[*num_of_entries].level.integrity_level);

          (*num_of_entries)++;

        }//endif add_to_list

        keep_going = 1;
     } //end else if (single)

     else if (!strcmp(choice_str,"range"))//if the menu selection is
range
     {
       printf("\n   Not implemented yet, try again");
       keep_going = 1;
     } //end else if (range)

     else if (!strcmp(choice_str,"list"))//if the menu selection is
range
     {
       keep_going = 1;
     } //end else if (list)

     else if (!strcmp(choice_str,"display"))//if the menu selection is
range
     {
       printf("\n   Not implemented yet, try again\n");
       keep_going = 1;
     } //end else if (display)

     else if (!strcmp(choice_str,"make"))//if the menu selection is
done
     {
       printf("\nList is complete  ");
       keep_going = -1;
     } //end else if (make)

     else if (!strcmp(choice_str,"exit"))//if the menu selection is
done
     {
       printf("\nNo mailboxes will be made\n");
       list = NULL;
       *num_of_entries = 0;
       keep_going = -1;
     } //end else if (exit)

     else
     {
        printf("\nERROR : Input not valid, try again.\n");
        keep_going = -1;
     } //end else

     if (keep_going == -1)
     {
        //nothing
     }
```

102

```c
      else
      {
          int ix = 0;
          printf("the current list of mailboxes to be made:\n");
          while(ix < (*num_of_entries))
          {
              printf("%s\n", list[ix].label);
              ix++;
          }//end while
          keep_going = request_yes_no("\nDo you want to edit the list?");
      } //end if else

   }//end while (keep going)

   return(list);
}
//------------END  Get_User_MAC_level------------------


//-------------------------------------------------------
//-- PROCEDURE     : void Get_User_MAC_label
//-- PURPOSE       : Gets the MAC label that corresponds to
//--                     the MAC level from the security map
//-- Pre           : MAC_level passed in
//-- Post          : An alias label is associated with the
//--                     MAC level
//-- Exceptions    : label doesnt exist
//-- Return        : void
//-- Parameters    : access_ma level, char* label
//-- -------------------------------------------------------

void Get_MAC_label(access_ma* level, char* label)
{

    init_hrl_db_for_access();

        if (get_hrl_db_label(level,
            label )==HRL_FOUND)
        {

            printf("label :  %s\n", label);

        }//endif
        else
        {
            printf("ERROR: label not found for this MAC level\n");
            label = NULL;
         }

        printf("***MAC level sl: %d il: %d\n",
                level->security_level,
                level->integrity_level);

}
//------------END Get_MAC_label-------------------------
```

103

```
//-----------------------------------------------------------
//-- PROCEDURE     : void Make_MAC_Directory
//-- PURPOSE       : Makes the MAC level directory
//--                        /home/username/mac_level
//-- Pre           : Parameters defined
//-- Post          : Directory created
//-- Exceptions    : directory already exists
//-- Return        : void
//-- Parameters    : char* user_directory, char* label,
//--                     access_ma* mac, char* MAC_directory
//-- -------------------------------------------------------

void Make_MAC_Directory(const char* user_directory,
                        const char* label,
                        access_ma* mac,
                        char* MAC_directory)
{

printf("***Making MAC label directory\n");

    error_code err;
    access_da dac ;
    dac.owner_perms =READ_WRITE_EXECUTE_MODE;
    dac.group_perms = 0;
    dac.other_perms =0;
    access access_level;
    access_level.ma = *mac;
    access_ma tempmac;
    int fd;//file descriptor
    sprintf(MAC_directory, "%s/%s", user_directory, label);


    printf("Making directory -   %s\n",MAC_directory);
    err = create_directory(MAC_directory,dac);//maked dir w/ rwx for
owner
    if (err != NO_ERROR)
    {
    print_error("***Creation of directory failed ", err);
    }
    //printf("***Exiting Make_MAC_Directory\n");//take out

    fd = open(MAC_directory, O_RDONLY);
    printf("*** file descriptor is %d\n", fd);
    if (fd == -1)
    {
        print_error("*** ERROR in open ", tcb_error_code);
    }
    else
    {
        err = get_fd_access(fd, &dac, &tempmac);
        if (err != NO_ERROR)
        {
        print_error("***Get FD access error  ", err);
        }
        else
        {
```

104

```
        access_level.da = dac;
        err= set_fd_access(fd, access_level);

        if (err != NO_ERROR)
        {
        print_error("***Change MAC Level of directory failed ", err);
        }
        close(fd);
        }
    }
}
//------------END Make_MAC_Directory--------------------


//----------------------------------------------------
//-- PROCEDURE     : void Create_mbox
//-- PURPOSE       : call IMAP to makes the mbox for a
//--                      user at a particular MAC level
//-- Pre           : Mac level box made
//-- Post          : mbox made
//-- Exceptions    : mbox already exists
//-- Return        : void
//-- Parameters    : access_ma* level
//-- ------------------------------------------------

void Create_mbox(access_ma* level)
{
error_code  the_err;
ushort      original_priv = 0;
procuid     child_proc;

const char* my_aps_path = "/usr2/shifflet/wip/imap/imap-4.6.BETA/imap-
utils/mbxcreat";


//original_priv = enable_priv();

printf("***Making mbox\n");

the_err = load_process(my_aps_path, *level, FALSE, NULL,
                       (&child_proc));

    if (the_err != NO_ERROR)
    {
       print_error("***ERROR load_process  ",the_err);
    }
    else
    {
       the_err = send_ipc_message(child_proc, 2, TRUE,
                                  DEVICE_AVAILABLE_EVENT,
                                  NULL, 0);
       print_error("***ERROR send_ipc_message  ",the_err);
    }
//set_priv(original_priv);
}
//------------END Create_mbox --------------------------
//
```

```
// File:     toolbox.h
// Author:   Richard K. Rossetti
// Date:     Sep 2000
// Purpose: Provide a tool to set up a mailbox
//          hierarchy on the XTS-300
#ifndef TOOLBOX_H
#define TOOLBOX_H
#ifdef OSS_OPTION
#include <user_group.h>
#endif
#include <stdio.h>
#include <sys/types.h>
#include <pwd.h>
#include "hrl_db.h"

typedef struct
{
  char label[HRL_LABEL_LEN+1];
  access_ma  level;
}MAC_alias_list;


//----------------------------------------------------
//-- PROCEDURE    : void Get_User_ID
//-- PURPOSE      : Gets the username and User ID
//--                       for a mailbox
//-- Pre          : Trusted process needs privilege to
//--                       "read down" to lower integrity
//--                       database
//-- Post         : User name and User Id initialized
//-- Exceptions   : Invalid input from the keyboard
//-- Return       : void
//-- Parameters   : char* name, short int &id
//-- ------------------------------------------------

void Get_User_ID(char* name, short int &id);

//----------------------------------------------------
//-- PROCEDURE    : void Get_Mail_Home
//-- PURPOSE      : Get the mail home          .
//-- Pre          : Default mail home defined
//-- Post         : mail home directory defined
//-- Exceptions   : Invalid input from the keyboard
//--                       string too long
//-- Return       : void
//-- Parameters   : char* home
//-- ------------------------------------------------

void Get_Mail_Home(char* home);

//----------------------------------------------------
//-- PROCEDURE    : void Make_User_Directory
//-- PURPOSE      : makes the directory /home/username
//-- Pre          : mail home defined, user defined
//-- Post         : a directory is created
//-- Exceptions   : directory already exists
```

106

```
//-- Return        : void
//-- Parameters    : const char* home, const char* name,
//--                    char* directory
//-- ------------------------------------------------------

void Make_User_Directory(const char* home,
              const char* name, char* directory);


//----------------------------------------------------------
//-- PROCEDURE      : void Get_MAC_list
//-- PURPOSE        : Gets a list of MAC labels and MAC
//--                    label aliases
//-- Pre            : user id defined
//-- Post           : A list of MAC labels and alias labels
//-- Exceptions     : Invalid user input
//-- Return         : MAC_alias_list struct
//-- Parameters     : short int id, *num_of_entries
//-- ------------------------------------------------------

MAC_alias_list*  Get_MAC_list(short int id,
                     int *num_of_entries);


//----------------------------------------------------------
//-- PROCEDURE      : void Get_User_MAC_label
//-- PURPOSE        : Gets the MAC label that corresponds to
//--                    the MAC level from the security map
//-- Pre            : MAC_level passed in
//-- Post           : A label is associated with the MAC level
//-- Exceptions     : label doesnt exist
//-- Return         : char string
//-- Parameters     : access_ma level, char* label
//-- ------------------------------------------------------

void Get_MAC_label(access_ma* level, char* label);

//----------------------------------------------------------
//-- PROCEDURE      : void Make_MAC_Directory
//-- PURPOSE        : Makes the MAC level directory
//--                    /home/username/mac_level
//-- Pre            : Parameters defined
//-- Post           : Directory created
//-- Exceptions     : directory already exists
//-- Return         : void
//-- Parameters     : char user_directory, char label,
//--                    access_ma* mac, char MAC_directory
//-- ------------------------------------------------------

void Make_MAC_Directory(const char* user_directory,
                   const char* label, access_ma* mac,
                   char* MAC_directory);

//----------------------------------------------------------
//-- PROCEDURE      : void  Upgrade_MAC_level
//-- PURPOSE        : Upgrades the MAC level of the mailbox
//-- Pre         : Mailbox made ant min max
//-- Post        : Mailbox at appropriate MAC level
//-- Exceptions : none
```

107

```
//-- Return      : void
//-- Parameters :  access_ma level,
//--                cnst char* MAC_directory.
//-- -----------------------------------------------------
//
//void Upgrade_MAC_level(access_ma* level,
//                     const char* MAC_dirirectory);
//
//-----------------------------------------------------
//-- PROCEDURE     : void  Change_owner
//-- PURPOSE       : Changes owner of mailbox to username
//-- Pre         : Mailbox owned by admin
//-- Post        : Mailbox owned by user rwx --- ---
//-- Exceptions : none
//-- Return      : void
//-- Parameters :  char name, char user_directory,
//--                char mailbox_name, char MAC_directory.
//-- -----------------------------------------------------
//
//void Change_owner(const char* name, const char* user_directory,
//                const char* MAC_directory);
//
//-----------------------------------------------------
//-- PROCEDURE     : void Create_mbox
//-- PURPOSE       : call IMAP to makes the mbox for a
//--                user at a particular MAC level
//-- Pre         : Mac level box made
//-- Post        : mbox made
//-- Exceptions  : mbox already exists
//-- Return      : void
//-- Parameters  : access_ma* level
//-- -----------------------------------------------------

void Create_mbox(access_ma* level);
#endif
```

LIST OF REFERENCES

1. Wood, David, *Programming Internet Email*, O'Reilly & Associates, Inc, August 1999.

2. *Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*, Rep No. CSC-STD-004-85, 25 June 1985.*Department of Defense Trusted Computer Systems Evaluation Criteria*, DoD 5200.28-STD, National Computer Security Center, December 1985.

3. *Department of Defense Trusted Computer Systems Evaluation Criteria*, DoD 5200.28-STD, National Computer Security Center, December 1985.

4. Downey, Robb, "Design of a High Assurance, Multilevel Secure Mail Server," Masters Thesis, Naval Postgraduate School, Monterey, CA, September 1997.Irvine, "A Multilevel File System for High Assurance," *In Proceeding of the IEEE Symposium on Security and Privacy, pages 78-87*, Oakland, CA, May 1995.

5. Irvine, Anderson, Robb, Hackerson, "High Assurance Multilevel Services For Off-The-Shelf Workstation Applications," *In Proceedings 21st National Information Systems Security Conference, pages 421-431*, Crystal City, VA, October 1998.

6. Wilson, "Trusted Networking in a Multilevel Secure Environment," Masters Thesis, Naval Postgraduate School, Monterey, California, June 2000.

7. Saltzer, Schroeder, "The Protection of Information in Computer Systems," *In Proceedings of the IEEE, Vol. 63, No. 9, pages 1278-1308*, September 1975.

8. Eads, "Developing A High Assurance Multilevel Mail Server," Masters Thesis, Naval Postgraduate School, Monterey, CA , March 1999.

9. Bell, La Padula, "Computer Security Model: Unified Exposition and Multics Interpretation," Technical Report ESD-TR-75-306, The MITRE Corporation, Bedford, MA, June 1975.

10. Biba, "Integrity Considerations for Secure Computer Systems," Technical report ESD-TR-76-372, The MITRE Corporation, Bedford, MA, April 1977.

11. *Final Evaluation Report, Wang Government Services Incorporated XTS-300*, Rep No. CSC-EPL-92/003.B, National Computer Security Center, July 1995.

12. *Final Evaluation Report, Wang Government Services Incorporated XTS-300*, Rep No. CSC-EPL-92/003.C, National Computer Security Center, April 1999.

13. *XTS-300 Application's Programmer's Reference Manual*, Document ID: FS92-374-06, Wang Government Services, Inc., March 1998.

14. *XTS-300 Trusted Facility Manual*, Document ID: FS92-371-07, Wang Government Services, Inc., March 1998.

15. *XTS-300 Trusted Programmer's Reference Manual*, Document ID: FS92-375-07, Wang Government Services, Inc., March 1998.

16. *XTS-300 Users Manual*, Document ID: FS92-373-07, Wang Government Services, Inc., March 1998.

17. Crispin, "Internet Message Access Protocol – Version 4 Revision 1, Internet-Draft," *RFC 2026 IMAP4rev1,* February 2000, URL: http://www.ietf.org/internet-drafts/draft-crispin-imap-09.txt (27 Jul 2000).

18. Meyers, Rose, "Post Office Protocol- Version 3" *RFC 1939 POP3,* May 1996.

19. Postel, "Simple Mail Transfer Protocol," *RFC 821 SMTP*, August 1982.

20. Grey,"Comparing Two Approaches to Remote Mailbox Access: IMAP vs. POP," The IMAP Connection, University of Washington, September 1995,URL: http://www.imap.org/imap.vs.pop.brief.html (20 Mar 2000).

21. *Glossary of Computer Security Terms*, Rep No. NCSC-TG-004-88, National Computer Security Center, October 1988.

BIBLIOGRAPHY

Clark, "A LINUX-Based Approach to low-cost Support of Access Control Policies," Masters Thesis, Naval Postgraduate School, Monterey, California, September 1999.

Internet Mail Consortium, URL: http://www.imc.org (7 Sep 2000).

Irvine, Acheson, Thompson, "Building Trust into a Multilevel File System," *In Proceedings 13th National Computer Security Conference, pages 450-459*, Washington, DC, October 1990.

Irvine, "A Multilevel File System for High Assurance," *In Proceeding of the IEEE Symposium on Security and Privacy, pages 78-87*, Oakland, CA, May 1995.

Korb. "Research Writing in Computer Science," Technical Report 97/308, Dept of Computer Science, Monash University, Clayton, Victoria, Australia, 1977.

Kramer, "Linus IV- An Experiment in Computer Security," *In Proceedings IEEE Symposium on Security and Privacy, pages 24-32*, April 1984.

Lampson, "Protection, " *In Proceedings 5th Princeton Conference on Information Sciences and Systems, page 437*, Princeton, 1971.

Storm, Rose, *Internet Messaging*, Prentice Hall, 1998, ISBN 0139786104.

Wang Government Services Website, URL: http://www.wangfed.com/security/ssso/gov_services_ssso_xts_page2_c.asp (22 Mar 2000).

Whitmore, Bensoussan, Green, Hunt, Kobziar, Stern, *Design for Multics Security Enhancements*, Technical Report ESD-TR-74-176, Hanscom AFB, MA.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

No. copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS 1
   Naval Postgraduate School
   Monterey, CA 93943-5101

4. Dr. Cynthia E. Irvine..3
   Computer Science Department, Code CS/Ic
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Mr. James P. Anderson 1
   James P. Anderson Company
   Box 42
   Fort Washington, PA

6. Paul C. Clark...1
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. Richard Kip Rossetti 2
   410 Warley St.
   Portsmouth, RI 02871

8. Mr. Paul Pitelli..1
   National Security Agency
   Research and Development Building
   R2, Technical Director
   9800 Savage Road
   Fort Meade, MD 20755-6000

2511 South Jefferson Davis Highway
Arlington, VA 22202

17. Mr. James Knoke 1
Wang Government Services Inc.
7900 Westport Dr.
Mclean, VA 22102-4299

18. Mr. Michael Focke..1
Wang Government Services Inc.
7900 Westport Dr.
Mclean, VA 22102-4299